PROFUNDIZANDO EN EL ZX SPECTRUM



Dilwyn Jones

EDITORIAL NORAY

PROFUNDIZANDO EN EL ZX SPECTRUM

Dilwyn Jones

EDITORIAL NORAY, S.A.
San Gervasio de Cassolas, 79
Tel. 211 11 46 - 08022 Barcelona

Cualquier duda o aclaración sobre esta obra, será contestada por el departamento técnico de Editorial Noray, siempre y cuando se solicite por escrito al Apartado de correos n.º 6015, 08080 de Barcelona.

Título original: Delwing deeper into your ZX Spectrum
Traducción de: Angeles Nogué - Carlos Cervera

Primera edición, 1985
Depósito Legal: B. 26.648-1985
ISBN: 84-7486-043-1
Número de edición de E.N., 67
Printed in Spain - Impreso en España
Gráficas Instar. C/ Industria, s/n Hospitalet del Llobregat (Barcelona)

ÍNDICE

Pág.

INTRODUCCIÓN

Bienvenidos, "Profundizadores"

Muchas de las preguntas y problemas que he encontrado durante mi experiencia con el ZX Spectrum me han servido de materia para realizar este libro. La cantidad de trucos y astucias que aparecen ayudarán al programador a obtener un conocimiento más profundo sobre el micro ZX Spectrum, incluso más allá de lo que le parecía inimaginable.

Espero que encuentre que los temas y las materias más avanzadas han sido escritos en un estilo simple de entender. El Spectrum es tan fácil de usar comparado con otros microcomputadores, que creo que los libros como éste deberían hacer lo posible para reflejarlo de la manera más familiar y accesible.

Con esta obra, espero que pueda profundizar desde la ROM hasta los programas en tres dimensiones.

Adelante pués y..... ¡a fondo!.

TRUCOS DE LA PANTALLA

Entre y ejecute este programa. ¿Qué es lo que hace?

```
  10 DIM i$(704)
  20 PRINT AT RND*20,RND*31;CHR$
 (RND*223+32)
  30 PRINT AT 0,0; OVER 1; INVER
SE 1;i$
  40 GO TO 20
```

En un par de segundos imprime algo en la pantalla y a continuación toda ella queda invertida. ¿Quién necesita el código de máquina?. Realmente, se hace imprimiendo espacios toda una pantalla OVER (sobre) la pantalla en INVERSE (inverso) la cual tiene el efecto de hacer que todo lo que era blanco en la pantalla se convierte en negro y que todo lo que era negro se convierte en blanco; normalmente Vd. esperaba que el OVER usaría su acción EXOR para borrar algunas partes pero el EXOR no tiene nada que borrar en una cadena de espacios de manera que suministra una pantalla en inverso muy rápidamente. Funciona bien en negro y blanco y es fácil hacer que trabaje en color añadiendo los controles de color PAPER (PAPEL), INK (TINTA), FLASH (destello) y BRIGHT (BRILLO) con un parámetro de 8 cada uno de ellos para prevenir la unión de ciertos colores si fuesen diferentes. Todo lo que esto hace es asegurar los mismos atributos mantenidos aunque el INVERSE 1 quede afectado.

```
  10 DIM i$(704)
  15 FOR i=1 TO 50
  20 PRINT AT RND*20,RND*31; INK
 RND*7; PAPER RND*7; FLASH RND;
BRIGHT RND;CHR$ (RND*223+32)
  30 PRINT AT 0,0; INVERSE 1; OV
ER 1; PAPER 8; INK 8; BRIGHT 8;
FLASH 8;i$
  40 NEXT i
```

La misma idea puede utilizarse para convertir todo el texto y gráficos de la pantalla en un color determinado omitiendo la sentencia INVERSE 1 (o especificando el INVERSE 0) y especificando un color INK en vez de dejarlo en INK8. Por ejemplo, este programa escribe los caracteres en la pantalla en colores INK y PAPER aleatorios, para demostración, entonces cambia todos los caracteres a negro mientras mantiene los mismos atributos de brillo, destello y color de fondo:

```
10 DIM i$(704)
15 FOR i=1 TO 50
20 PRINT AT RND*20,RND*31; INK
```

```
 RND*7; PAPER RND*7; FLASH RND;
BRIGHT RND;CHR$ (RND*223+32)
  30 PRINT AT 0,0; INVERSE 1; OV
ER 1; PAPER 8; INK 0; BRIGHT 8;
FLASH 8;i$
  40 NEXT i
```

Habrá observado que algunos INKs y PAPERs son los mismos que aparecen después de la impresión aleatoria en la línea 20. Esto es un problema muy común. ¿Problema?. ¡No!. Tan sólo especifique el INK 9. Ahora Vd. puede leer todo.

```
  10 DIM i$(704)
  15 FOR i=1 TO 50
  20 PRINT AT RND*20,RND*31; INK
 RND*7; PAPER RND*7; FLASH RND;
BRIGHT RND;CHR$ (RND*223+32)
  30 PRINT AT 0,0; INVERSE 1; OV
ER 1; PAPER 8; INK 9; BRIGHT 8;
FLASH 8;i$
  40 NEXT i
```

Podemos hacer lo mismo para el PAPER. Especificando el color del PAPER, y dejando todos los otros atributos igual, todo el color del fondo puede cambiarse sin que afecte a lo que hay en la pantalla o usando el CLS. ¡Observe que todo el texto escrito en este color aparece en la pantalla como disipado pues el texto verde sobre fondo verde no es tan fácil de leer!. Este ejemplo dibuja los caracteres aleatorios con atributos aleatorios, entonces fija todo el fondo en amarillo.

```
  10 DIM i$(704)
  15 FOR i=1 TO 50
  20 PRINT AT RND*20,RND*31; INK
 RND*7; PAPER RND*7; FLASH RND;
BRIGHT RND;CHR$ (RND*223+32)
  30 PRINT AT 0,0; INVERSE 1; OV
ER 1; PAPER 8; INK 8; BRIGHT 8;
FLASH 8;i$
  40 NEXT i
```

Vd. puede obtener un efecto interesante con cualquier área que tenga un atributo BRIGHT de 1 con el programa de arriba. Si Vd. ha suministrado indicadores del usuario en el BRIGHT 1 o en el FLASH 1 (o sea, extra brillo o destello) para remarcarlos y después de conseguir lo que Vd. quería, cancelarlos, Vd. puede hacerlo de las formas siguientes:

Para desactivar el brillo:

```
   10 DIM i$(704)
   15 FOR i=1 TO 50
   20 PRINT AT RND*20,RND*31; INK
 RND*7; PAPER RND*7; FLASH RND;
BRIGHT RND;CHR$ (RND*223+32)
   30 PRINT AT 0,0; INVERSE 1; OV
ER 1; PAPER 8; INK 8; BRIGHT 0;
FLASH 8;i$
   40 NEXT i
```

Para desactivar el destello:

```
   10 DIM i$(704)
   15 FOR i=1 TO 50
   20 PRINT AT RND*20,RND*31; INK
 RND*7; PAPER RND*7; FLASH RND;
BRIGHT RND;CHR$ (RND*223+32)
   30 PRINT AT 0,0; INVERSE 1; OV
ER 1; PAPER 8; INK 8; BRIGHT 8;
FLASH 0;i$
   40 NEXT i
```

Observe que en todos los ejemplos de arriba, los "trucos de la pantalla" ¡se realizan en una sola línea!. Recuerde: la respuesta a la última pregunta sobre la vida, el universo y todo el resto es una cadena de 704 espacios impresos OVER 1 sobre toda la pantalla en el color   8.

Esta técnica abre una interesante posibilidad. Si Vd. quiere dibujar una forma compleja la cual iría muy despacio, primero dibújela de la forma normal en el mismo color con INK y PAPER de manera que quede invisible, entonces use la técnica de arriba para cambiar el color de la forma de manera que se haga visible casi instantáneamente. Intente este programa que dibuja cuatro círculos concéntricos en magenta sobre un fondo amarillo. El proceso de dibujo tarda unos 4 segundos.

```
 5 INK 3; PAPER 6: CLS
10 DIM i$(704)
15 FOR i=10 TO 70 STEP 20
20 CIRCLE 120,90,i
25 NEXT i
```

Pruebe con este programa el cual inicialmente dibuja los círculos en amarillo sobre un fondo amarillo entonces, después de dibujar, cambia el color de los círculos a magenta sobre amarillo. Durante unos segundos Vd. estará observando una pantalla amarilla sin nada, pero cuando los círculos aparecen lo hacen de forma casi instantánea. En la práctica, Vd. podrá disimular el retraso de manera que el dibujo aparezca instantáneamente.

```
   5 INK 6: PAPER 6: CLS
  10 DIM i$(704)
  15 FOR i=10 TO 70 STEP 20
  20 CIRCLE 120,90,i
  25 NEXT i
  30 PRINT AT 0,0; INK 3; OVER 1
;i$
```

Esto sólo es el esqueleto de una idea, pero usando una cadena de sobreimpresión de espacios para controlar los atributos del fichero es una herramienta de programación rápida y potente.

Hemos estado hablando en términos de como usar una pantalla llena de espacios, para afectar la pantalla entera. Vd. puede usar lo justo para alterar los atributos de un solo carácter o palabra en la pantalla, es decir, para hacer que el monstruo verde se convierta en blanco de terror cuando Vd. le acierta con su arma espacial:

```
PRINT AT Y,X; OVER 1; INK 7;"*"
```

o para hacer un ciclo de palabras con todos los atributos posibles:

```
  10 PRINT AT 5,5;"Socorro"
  20 FOR f=0 TO 1: FOR b=0 TO 1:
FOR p=0 TO 7: FOR i=0 TO 7
  30 PRINT AT 5,5; OVER 1; FLASH
 f; BRIGHT b; PAPER p; INK i;"
  ": REM 4 Espacios
  40 NEXT i: NEXT p: NEXT b: NEX
T f
```

ESCAPANDO DE LAS ENTRADAS

Si Vd. desea parar un programa durante un INPUT, el BREAK no opera y tan sólo produce un espacio a añadir al INPUT. En el caso de un INPUT numérico como el INPUT A en la respuesta debe escribirse STOP (que es el símbolo shift A) seguido por ENTER. El programa termina con el reporte H STOP en el INPUT.

En el caso de las cadenas, las cosas son diferentes. El STOP todavía puede utilizarse, pero el cursor debe ser la primera cosa en la línea y esto significa mover el cursor fuera de las comillas tanto si es con DELETE (que es CAPS SHIFT 0) o con CURSOR LEFT (que es CAPS SHIFT 5) entonces escriba STOP seguido de ENTER y el programa terminará con el reporte H STOP en el INPUT.

El uso de la facilidad INPUT LINE puede ser problemático a este respecto. El STOP se acepta como un carácter de entrada perfectamente válido y no detiene el programa. Sin embargo, es posible escapar del INPUT LINE usando CURSOR DOWN (que es CAPS SHIFT 6). Vd. no necesita pulsar el ENTER.¡Otra vez el programa terminará con el reporte H STOP en el INPUT a pesar de que el STOP no se ha utilizado!.

SINGULARES Y PLURALES

Un error común de un programa es terminar con una sentencia como "Queda 1 bombas", o sea, no dar importancia a los singulares y plurales. La línea del programa que genera esta sentencia es algo así:

```
1000 PRINT "Todavia quedan ";bom
bas;" bombas"
```

Todo lo que está mal es que el programa no puede con la gramática del lenguaje Español. En este contexto, es posible acercarse al problema muy fácilmente. Aquí hay una forma de acercarse al problema:

```
1000 PRINT "Todavia ";
1010 IF bombas>1 THEN PRINT "que
dan";
1020 IF bombas<=1 THEN PRINT "qu
eda";
1030 PRINT " ";bombas;" bomba";
1040 IF bombas>1 THEN PRINT "s";
```

Quizás la rutina sea algo larga y embarazosa para una tarea tan simple. Esta rutina usa el AND para acortarla a una sola línea

```
1000 PRINT "Todavia ";"quedan" A
ND bombas>1;"queda" AND bombas<=
1;" ";bombas;" bomba";"s" AND bo
mbas>1
```

PARA PERMITIR QUE EL SPECTRUM ACTIVE/DESACTIVE EL CAPS LOCK

El bit 3 de la variable 23658 del sistema (FLAGS2) indica el estado del CAPS LOCK. Si está activado el bit 3 es 1, si no está activado el bit 3 es 0. En los programas, a menudo es útil poder detectar cuando una cierta tecla se pulsa independientemente de que el carácter pulsado está en mayúsculas o minúsculas, es decir, para una respuesta SI o NO. Para activar el CAPS LOCK use la sentencia POKE 23658,8 y para desactivar el CAPS LOCK use el POKE 23658,0 teniendo en cuenta que éstos afectarán los otros flags de la variable del sistema. Para una demostración divertida, conecte la impresora ZX y entre esto:

POKE23658,2

Vd. no dañará la impresora o perderá papel o nada de todo esto, pero la despertará. Así, Vd. puede ver que es necesario poner atención cuando haga un POKE en la posición 23658. Así es como el SI o NO de una rutina puede trabajar:

```
1000 PRINT "Desea otra partida (
S o N)?"
1010 POKE 23658,8
1020 IF INKEY$="S" THEN RUN
1030 IF INKEY$="N" THEN STOP
1040 GO TO 1020
```

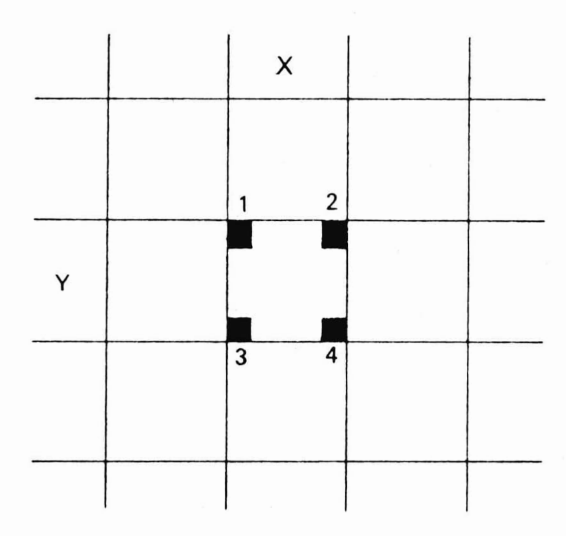
Para no modificar los otros flags
Activ: POKE 23658, (PEEK (23658) OR 8)
Desact: POKE 23658, (PEEK (23658) AND 247)

PAUSA Y BUCLES FOR/NEXT

Normalmente no hay ningún problema al usar el PAUSE en el Spectrum pero cuando se necesita un retraso fijo, el PAUSE puede causar problemas. El PAUSE se desactiva con la pulsación de una tecla, así pues, si Vd. pulsa una tecla no sucederá ningún PAUSE. Este problema puede evitarse mediante el uso de los bucles FOR/NEXT como bucles de espera. Para realizar una pausa de 1 segundo use un bucle de FOR A = 1 TO 220:NEXT A

PARA RELACIONAR LAS COORDENADAS DE LAS SENTENCIAS PRINT Y PLOT

Supongamos que Vd. tiene PRINT AT Y,X; la posición Y hace filas verticales y la columna X a través de la pantalla. Y sería 0 en la parte superior de la pantalla y 21 en la parte inferior. X sería 0 a la izquierda de la pantalla y 31 a la derecha de la pantalla; en otras palabras, las coordenadas PRINT estándard.



Las coordenadas de dibujo que corresponden a los cuatro rincones 1 al 4 de la celda de caracteres Y,X (mostrado en el diagrama de arriba) estarían en el formato PLOT X,Y:

(1) X * 8, (21 – Y) * 8 + 7
(2) X * 8 + 7, (21 – Y) * 8 + 7
(3) X * 8, (21 – Y) * 8
(4) X * 8 + 7, (21 – Y) * 8

Con lo indicado arriba, Vd. debe de ser capaz de trabajar en todas las posiciones de pixels dentro de una posición PRINT en el caso de que Vd. necesite efectuar un PLOT o un DRAW sobre una posición PRINT conocida.

COMO HACER UN CATALOGO DE LOS PROGRAMAS EN UNA CINTA

Si Vd. tiene una cinta llena de programas y no sabe sus nombres o naturaleza (es decir, los programas en BASIC, los registros de datos o los bytes de memoria) le sería de ayuda ejecutarlos en la cinta y tener los nombres escritos en la pantalla de TV con la cinta del registro, sin afectar para nada lo que se ha entrado recientemente en la memoria del computador. Los tipos de registros que Vd. puede encontrarse así son:

Matriz alfanumérica: A Matriz numérica: A	} para SAVE"A" DATA A() o A$()
Programa: Laberinto	para SAVE Laberinto
Bytes: código	para SAVE"código" CODE 16384,32

Para hacer esto, use el VERIFY con el nombre de un programa que nunca se encontrará en la cinta. Yo intento usar VERIFY"ZZZZZZZZZZ". Asegúrese de que no hay ningún otro nombre de programa en la pantalla, ya que puede confundirle.

BORRADO PARCIAL DE LA PANTALLA

El INPUT también puede utilizarse sin tener que entrar realmente una variable. Esto puede utilizarse para hacer un borrado parcial de la parte inferior de la pantalla siempre que el INPUT AT no sobrepase la posición principal de PRINT (en cuyo caso la pantalla empieza a deslizarse). Use INPUT AT así, recordando que las coordenadas del INPUT AT comienzan desde la parte inferior de la pantalla.

```
  10 INPUT "¿Donde?",W
  20 FOR A=0 TO 21: PRINT A: NEX
T A
  30 PRINT AT 0,0;
  40 INPUT AT W,0;
```

Si el INPUT AT llega a la actual posición de PRINT la pantalla se deslizará. Vd. podría guardar la posición de impresión, moverla a otro lugar, borrar la parte inferior de la pantalla y entonces volver a colocar la posición de impresión.

```
  10 FOR A=0 TO 21: PRINT AT A,0
;A;: NEXT A
  20 LET X=33-PEEK 23688
  30 LET Y=24-PEEK 23689
  40 PRINT AT 0,0;
  50 INPUT "¿Cuántas?",HM
  60 INPUT AT HM+1,0;
  70 PRINT AT Y,X;
```

En la rutina de arriba, la línea 10 imprime algo en la pantalla. Las líneas 20 y 30 guardan las coordenadas de la posición PRINT. La línea 40 mueve la posición de impresión temporalmente hasta la parte superior de la pantalla. La línea 50 pregunta cuántas líneas quiere borrar de las 22 líneas de la parte superior de la pantalla. Por ejemplo, entrando 1, Vd. sólo borraría la línea 21. La línea 60 es la que efectúa el borrado, y la línea 70 recupera la posición de impresión.

DESLIZAMIENTO DE LA PANTALLA

Es posible crear un efecto de deslizamiento desde el BASIC almacenando la imagen de la pantalla en una cadena lo suficientemente larga como para contener todos los 22 * 32 caracteres de la pantalla. El deslizamiento se crea por la rotación de los elementos de la cadena. El deslizamiento hacia arriba y abajo puede realizarse de esta manera bastante rápido:

```
 1 REM Deslizar Arriba
10 DIM a$(704)
20 INPUT a$
30 PRINT AT 21,0,0;a$
40 LET a$=a$(33 TO )+"
                              "
50 GO TO 30
```

```
 1 REM Deslizar Abajo
10 DIM a$(704)
20 INPUT a$
30 PRINT AT 0,0;a$
40 LET a$="
                   "+a$( TO 672)
50 GO TO 30
```

Los deslizamientos laterales son más lentos, pero posibles:

```
 1 REM Deslizar Izquierda
10 DIM a$(704)
20 INPUT a$
30 PRINT AT 0,0;a$
40 FOR f=1 TO 673 STEP 32
50 LET a$(f TO f+31)=a$(f+1 TO
f+31)+" "
60 NEXT f
70 GO TO 30
```

```
 1 REM Deslizar Derecha
10 DIM a$(704)
20 INPUT a$
30 PRINT AT 0,0;a$
40 FOR f=1 TO 673 STEP 32
50 LET a$(f TO f+31)=" "+a$(f
TO f+30)
60 NEXT f
70 GO TO 30
```

EL CONJUNTO DE CARACTERES

En esta sección veremos todos los posibles usos del conjunto de caracteres de la ROM, después procederemos a mover el conjunto de caracteres de la ROM a la RAM (figuradamente hablando) y hacer nuestros propios alfabetos. Pero primero, estudiemos una importante variable del sistema, pruebe lo siguiente: Entre como un comando directo:

POKE23606,8

entonces pulse ENTER, naturalmente. Ahora intente escribir en un programa ¡si puede!. Todavía más divertido, pruebe de escribir este corto programa. Reinicialice la máquina y escriba:

```
  10 POKE 23606,8
  20 PRINT INKEY$;
  30 IF INKEY$=" " THEN POKE 236
06,0: STOP
  40 GO TO 20
```

Ejecute (RUN) el programa y escriba algo –¿qué sucede?. Las letras A aparecen como B, las B como C y así sucesivamente. Quizás desee demostrar esto en su club de Spectrum, entonces dígales que el Spectrum se ha roto. Realmente, si Vd. pulsa la tecla SPACE (sin el SHIFT) el programa reinicializa las cosas. Todavía más divertido, quizás Vd. quiera cambiar la línea 10 por:

```
10 POKE 23606,4
```

¡para medio confundirse Vd. mismo!.

Explicación: 23606/23607 es la dirección de la variable del sistema que le dice al computador donde se encuentran las listas de los datos requeridos para la visualización de los caracteres en la pantalla. Esta variable del sistema de dos bytes contiene un número, 15360, después de conectarla que es 256 menos donde esta tabla comienza en la ROM. Así pues, 15360 + 256 = 15616. Aquí es donde comienza el GENERADOR DE CARACTERES DE LA ROM. Hagamos un PEEK allí para ver lo que hay. Entre y haga un RUN de este programa. Debería obtener resultados similares a la ejecución del ejemplo que sigue a continuación:

```
10 FOR A=15616 TO 15639
20 PRINT PEEK A
30 NEXT A
```

```
0
0
0
0
0
0
0
16
16
16
16
0
16
0
0
36
36
0
0
0
0
0
```

No fue demasiado informativo, ¿no?. El 15639 no era significativo, tan solo un número escogido para proporcionar un corto listado del ejemplo. Aunque miremos el conjunto de caracteres en el manual no parece tener ninguna relevancia para los números que se imprimen. ¡Ah,de acuerdo!, me olvidaba de otro requisito.

Intentémoslo en binario, ya que se supone que las computadoras hacen un gran uso de ello. Escriba y haga un RUN de este programa, teniendo en cuenta que los dos apóstrofes en la línea 70 son bastante fáciles de olvidar.

```
20>LET P=PEEK A
30 FOR B=7 TO 0 STEP -1
40 PRINT AT 21,B;P-2*INT (P/2)
50 LET P=INT (P/2)
60 NEXT B
70 PRINT AT 21,12;A''
80 NEXT A
```

```
00000000    15616
00000000    15617
00000000    15618
00000000    15619
00000000    15620
00000000    15621
00000000    15622
00000000    15623
00000000    15624
00010000    15625
00010000    15626
00010000    15627
```

```
00010000    15628
00000000    15629
00010000    15630
00000000    15631
00000000    15632
00100100    15633
00100100    15634
00000000    15635
00000000    15636
00000000    15637
00000000    15638
00000000    15639
```

No parece que hayamos ganado nada, sin embargo, intente imaginar la columna izquierda de la impresión sin los ceros así:

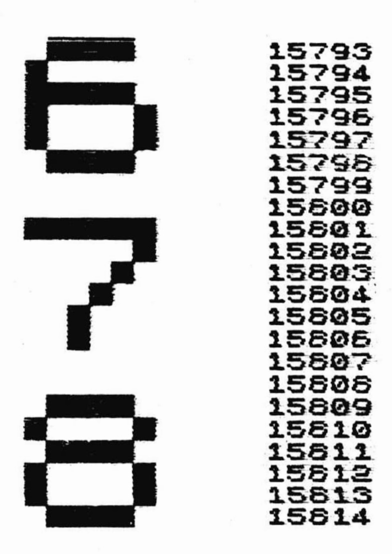
```
            15617
            15618
            15619
            15620
            15621
            15622
            15623
            15624
   1        15625
   1        15626
   1        15627
   1        15628
            15629
   1        15630
            15631
            15632
  1  1      15633
  1  1      15634
            15635
            15636
            15637
            15638
```

Lo que debería ver es un ESPACIO, un signo de admiración y algún otro símbolo (si es que Vd. es imaginativo). Para nuestro próximo truco escudriñaremos el generador de caracteres por entero de manera que veamos lo que está contenido en sus oscuras profundidades. Para apreciar mejor la imagen de la pantalla usaremos ■ en lugar de unos.

```
  10 FOR A=15616 TO 16383
  20 LET P=PEEK A
  30 FOR B=7 TO 0 STEP -1
  40 PRINT AT 21,B;"■" AND (P-2*
INT (P/2))=1
  50 LET P=INT (P/2)
  60 NEXT B
  70 PRINT AT 21,12;A''
  80 NEXT A
```

40 PRINT AT 21,9; PEEK A ; AT 21,B; "■" AND (P-2*INT (P/2))=1

Aquí hay un ejemplo de lo que obtendrá.



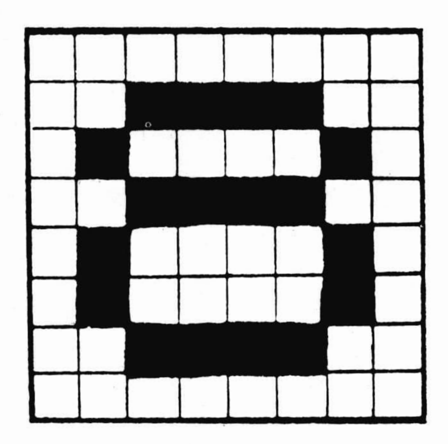
Si Vd. tiene una impresora y le gustaría guardar una copia sobre papel, puede usar este programa el cual le dará una larga impresión de todos los caracteres almacenados en el generador de caracteres de la ROM. Aviso: ¡necesitará una gran cantidad de papel para esto!. Pulse BREAK para detener la impresión cuando ya tenga suficiente.

```
  10 FOR A=15616 TO 16383
  20 LET P=PEEK A
  30 DIM A$(8)
  40 FOR B=8 TO 1 STEP -1
  50 IF P-INT (P/2)*2 THEN LET A
$(B)="█"
  60 LET P=INT (P/2)
  70 NEXT B
  80 LPRINT A$;TAB 10;A;TAB 18;P
EEK A
  90 NEXT A
```

A estas alturas debería coger la idea de que el generador de caracteres contiene un patrón de un bit por bit de lo que parecen los caracteres en la pantalla. También, si Vd. mira el Apéndice A del manual del Spectrum verá que todos los caracteres aparecen en el mismo orden en la pantalla igual a como lo hacen en el apéndice. Por lo menos los que están entre el rango de 32 a 127. El resto son los CARACTERES DE CONTROL (que en vez de aparecer en la panta-

lla la controlan), los caracteres gráficos (que están guardados en algún lugar), los gráficos de bloques (los cuales son "calculados" más que almacenados en patrones de bits) o están hechos de varias combinaciones de caracteres en el generador de caracteres (o sea, RUN, INKEY$, IF, etc.) en cuyo caso hay otra tabla en la ROM que indica las combinaciones.

No es ningún accidente el que estén por orden numérico. Están así deliberadamente para que al computador le sea más fácil encontrarlos cuando los necesite. Para comprender esto veamos como un carácter se organiza cuando mira a la pantalla, por ejemplo el número 8:



Un carácter está compuesto de hasta ocho por ocho ejes (dibujo de casillas o pequeños cuadrados si Vd. prefiere) en la pantalla. Por suerte parece ser de ocho bits por byte (¿¿¿puede creerlo???). Así pues, si cada byte de ocho bits representa una línea horizontal de caracteres, podríamos almacenarlos en un patrón de caracteres de ocho bytes. Así es como funciona el generador de caracteres: hay ocho bytes para cada carácter, almacenando el patrón como una serie de ceros y unos. Los unos representan las partes de los caracteres acoplados y los ceros representan las partes en blanco de los caracteres (en papel de color). Así es como aparecería el patrón de la figura 8 (la columna izquierda es el patrón del bit, la columna derecha muestra donde está colocado en decimal en la ROM):

```
00000000    15808   0
00111100    15809   60
01000010    15810   66
00111100    15811   60
01000010    15812   66
01000010    15813   66
00111100    15814   60
00000000    15815   0
```

Una utilidad que podemos añadir a esto es la de poder ampliar los caracteres. Si imprimimos un ■ para cada 1 podemos ampliar los caracteres ¡ocho veces!. Intentémoslo. El programa es un poco diferente de los que hemos utilizado hasta ahora, así pues, estúdielo cuidadosamente.




```
   10 LET atraves=0
   20 LET abajo=0
   30 INPUT a$
   40 LET c=CODE a$
   50 FOR k=0 TO 7
   60 LET p=PEEK (15360+c*8+k)
   70 FOR f=0 TO 7
   80 PRINT AT abajo+k,atraves+7-
f;"■" AND (p-2*INT (p/2)=1)
   90 LET p=INT (p/2)
  100 NEXT f
  110 NEXT k
  120 IF atraves+8>31 THEN LET ab
ajo=abajo+8
  130 LET atraves=atraves+8 AND a
traves+8<=31
  140 GO TO 30
```

Las dos variables *abajo* y *a través* controlan donde se imprime en la pantalla. A$ es el carácter que Vd. entra para ampliar. Debería consistir de un carácter con un CODE (CODIGO) de 32 a 127 (es decir, ESPACIO para el símbolo del copyright©). c es el CODIGO de este carácter. Observe cómo en la línea 60 se usa el número 15360 para el carácter situado en la parte inferior de la tabla del conjunto de caracteres. Vd. puede que recuerde que este número es 256 menos que la dirección del comienzo de la tabla. ¿Por qué?.

Bien, el primer patrón en la tabla es el del ESPACIO, que es CHR$(32). Recuerde también que hay ocho bytes para el patrón de puntos de cada carácter, de forma que todo comienza en múltiples de ocho. 8 veces 32 es 256 y 15360 + 256 es 15616, el comienzo de la tabla en la ROM. Los bucles van dividiendo por dos buscando el resto para determinar la forma en una parte de la pantalla. Entonces se ajustan los valores *abajo* y *a través*. Quizás Vd. prefiera entrar esta línea adicional por si el programa no puede mantener los caracteres que se entran (Vd. debe reentrar cualquier carácter que haya sido rehusado).

```
  35 IF CODE a$<32 OR CODE a$>12
7 THEN GO TO 30
```

Los caracteres que aparecen son muy grandes y Vd. no puede meter demasiados en la pantalla. A continuación viene cómo usar el PLOT y el DRAW para generar caracteres de varios tamaños.

```
   1 REM Caracteres
  10 INPUT ,"Veces mas ancho (1=n
ormal)?";másancho
  20 INPUT "Veces mas alto (1=no
rmal)?";masalto
  30 LET através=masancho*8-1: L
ET abajo=176
  40 INPUT a$: IF a$<"" OR a$>"©
" THEN GO TO 40
  50 FOR a=0 TO 7
  60 LET peek=PEEK (15360+CODE a
$*8+a)
  70 FOR b=0 TO 7
  80 IF peek-2*INT (peek/2) THEN
 FOR t=1 TO másalto: PLOT atrave
s-b*masancho,abajo-a*masalto-t:
DRAW 1-masancho,0: NEXT t
 100 LET peek=INT (peek/2): NEXT
 b
 110 NEXT a
 120 LET atraves=atraves+masanch
o*8
 130 IF atraves>255 AND abajo-ma
salto*8>masalto*8-1 THEN LET aba
jo=abajo-masalto*8: LET atraves=
masancho*8-1
 140 IF atraves>255 AND abajo-ma
salto*8<masalto*8 THEN PRINT AT
21,3': FOR a=1 TO masalto: PRINT
 : NEXT a: LET atraves=masancho*
8-1
 150 GO TO 40
```

El programa es complicado, de manera que estudie la siguiente información cuidadosamente. Cuando Vd. ejecute el programa,primero le preguntará cuánto más ancho de lo normal quiere que aparezca el carácter en la pantalla. Si

Vd. quiere que los caracteres sean tres veces más anchos que lo normal, entonces debería escribir 3, seguido de ENTER. Si quiere el ancho normal de los caracteres entre 1. Debería hacer lo mismo cuando se le pregunte cuántas veces quiere los caracteres más altos de lo normal. El programa comenzará en la parte superior de la pantalla e irá bajando a través hasta que alcance el final, cuando toda la pantalla se deslize arriba para hacer espacio a una nueva línea. El programa se ejecuta en color negro y blanco permanente (o los colores permanentes INK y PAPER que Vd. fijó) pero si lo desea puede añadir otros colores al programa.

Las variables *masalto* y *masancho* se utilizan para hacer que su significado quede claro. Lo mismo sucede con la variable *peek* (está escrita en minúsculas en el listado para que se distinga de la tecla PEEK) y las variables *atraves* y *abajo.* Existen coordenadas para los comandos PLOT que se usan más adelante. El 176 es uno más que el valor límite de 175 para PLOT – no se preocupe si hace errores de generación. *atraves* comienza a partir de una cierta cifra en la pantalla ya que el dibujar se hace de derecha a izquierda debido a la manera de evaluar el sistema binario. La línea 40 busca para ver qué tecla está pulsando. Este programa usa el INPUT para ver qué letras o símbolos quiere ampliar pues no existen demasiados símbolos con el INKEY$, que por otra parte le evita pulsar ENTER todo el tiempo. La línea 50 comienza el bucle que busca en el generador de caracteres los ocho bytes que necesita para ensamblar el carácter en la pantalla. Estos bytes se encuentran en la línea 60. El bucle que comienza en la línea 70 determina el patrón en su pantalla tomando el byte del generador de caracteres y repetidamente revisa para ver si los bits individuales están fijados de manera que áreas adecuadas puedan ser sombreadas en la pantalla. Esto se hace dibujando una línea,la cual es más *masancha* que un eje. Esta se hace más *alta* para que sea proporcional a su tamaño. La línea 100 divide el valor del *peek* por 2 de manera que el siguiente bit pueda ser revisado. La línea 120 fija el nuevo valor horizontal – si éste se sale del margen derecho de la pantalla, se da un nuevo valor a la variable *atraves* y el programa decide si es necesario el deslizamiento de la pantalla, el cual se hace mediante el PRINT bastantes veces. La variable *atraves* se reinicializa a su valor primitivo para una nueva línea de caracteres. Después de todo esto, la historia vuelve a repetirse otra vez.

El programa funciona mejor con los valores enteros de *masancho* y *masalto* pero Vd. también puede obtener buenos resultados para valores no enteros de *masancho* – aunque sea creando la ilusión de tener más o menos caracteres por línea coordinándolos con el valor de la variable *atraves*:

$$atraves = \frac{\text{caracteres normales en pantalla}}{\text{caracteres requeridos en pantalla}}$$

o sea, para cuarenta caracteres por línea, *através* sería 32/40 o 0.8. A continuación vienen algunos ejemplos producidos por el programa incluyendo uno con 40 caracteres por línea.



Utilizando el mismo principio podemos usar el programa para crear la imagen de los caracteres en un espejo:

Esto está hecho simplemente dibujando en la dirección opuesta:

```
   1 REM Espejos
  10 INPUT "Veces mas ancho (1=n
ormal)?";masancho
  20 INPUT "Veces mas alto (1=no
rmal)?";masalto
  30 LET atraves=0: LET abajo=17
5
  40 INPUT a$: IF a$<"" OR a$>"©
" THEN GO TO 40
  50 FOR a=0 TO 7
  60 LET peek=PEEK (15360+CODE a
$*8+a)
  70 FOR b=0 TO 7
  80 IF peek-2*INT (peek/2) THEN
 FOR t=1 TO masalto: PLOT atrave
s+b*masancho, abajo-a*masalto-t:
 DRAW 1-masancho,0: NEXT t
 100 LET peek=INT (peek/2): NEXT
 b
 110 NEXT a
 120 LET atraves=atraves+masanch
o*8
 130 IF atraves>256-masancho*8 A
ND abajo-masalto*8>masalto*8-1 T
HEN LET abajo=abajo-masalto*8: L
ET atraves=0
 140 IF atraves>256-masancho*8 A
ND abajo-masalto*8<masalto*8 THE
N PRINT AT 21,31': FOR a=1 TO ma
salto: PRINT : NEXT a: LET atrav
es=0
 150 GO TO 40
```

El siguiente paso es transformar el programa en una subrutina para usar en sus propios programas:

```
   1 REM Subrrutina
   4 LET masancho=2
   8 LET masalto=4
  12 LET atraves=35
  16 LET abajo=110
  20 LET a$="Demostracion"
  24 GO SUB 40
  30 STOP
  40 FOR d=1 TO LEN a$
  50 FOR a=0 TO 7
  60 LET peek=PEEK (15360+CODE a
$(d)*8+a)
```

```
  70 FOR b=0 TO 7
  80 IF peek-2*INT (peek/2) THEN
 FOR t=1 TO masalto: PLOT atrave
s-b*masancho,abajo-a*masalto-t:
DRAW 1-masancho,0: NEXT t
  90 LET peek=INT (peek/2): NEXT
b
 100 NEXT a
 110 LET atraves=atraves+masanch
o*8
 120 IF atraves>255 AND abajo-ma
salto*8>masalto*8-1 THEN LET aba
jo=abajo-masalto*8: LET atraves=
masancho*8-1
 130 IF atraves>255 AND abajo-ma
salto*8<masalto*8 THEN PRINT AT
21,31': FOR a=1 TO masalto: PRIN
T : NEXT a: LET atraves=masancho
*8-1
 140 NEXT d
 150 RETURN
```

Antes de llamar a la subrutina (que está situada en las líneas de la 40 a la 160) Vd. necesita especificar cuatro variables, *masancho, masalto, atraves* y *abajo,* igual como se utilizó en los otros programas. *atraves* y *abajo* necesitan especificarse como las coordenadas del PLOT de la parte superior del primer carácter. A$ es la cadena que contiene los caracteres a añadir y éstos se hacen uno por uno mediante el bucle extra D del bucle FOR/NEXT. También debería asegurarse de que A$ no contiene ningún carácter que no sea permitido o adicione esta línea al programa, la cual hará saltar el carácter no permitido (son los que tienen los CODIGOS menores que 32 y mayores de 127):

```
  45 IF a$(d)<" " OR a$(d)>"©" T
HEN GO TO 150
```

La subrutina también puede utilizarse con los gráficos definidos por el usuario mediante la adición de una línea de programa que determina cómo se deriva el valor del *peek.*

```
   1 REM Subrrutina
   4 LET masancho=2
   8 LET masalto=4
  12 LET atraves=35
  16 LET abajo=110
  20 LET a$="Demostración"
  24 GO SUB 40
  30 STOP
  40 FOR d=1 TO LEN a$
  45 IF a$(d)<" " OR a$(d)>"©" T
HEN GO TO 150
  50 FOR a=0 TO 7
  60 IF CODE a$(d)>31 AND CODE a
$(d)<128 THEN LET peek=PEEK (153
60+CODE a$(d)*8+a)
  61 IF CODE a$(d)>143 AND CODE
a$(d)<165 THEN LET peek=PEEK (US
```

```
R "a"+(CODE a$(d)-144)*8+a)
  70 FOR b=0 TO 7
  80 IF peek-2*INT (peek/2) THEN
 FOR t=1 TO masalto: PLOT atrave
s-b*masancho,abajo-a*masalto-t:
DRAW 1-masancho,0: NEXT t
  90 LET peek=INT (peek/2): NEXT
 b
 100 NEXT a
 110 LET atraves=atraves+masanch
o*8
 120 IF atraves>255 AND abajo-ma
salto*8>masalto*8-1 THEN LET aba
jo=abajo-masalto*8: LET atraves=
masancho*8-1
 130 IF atraves>255 AND abajo-ma
salto*8<masalto*8 THEN PRINT AT
21,31': FOR a=1 TO masalto: PRIN
T : NEXT a: LET atraves=masancho
*8-1
 140 NEXT d
 150 RETURN
```

Nuestro siguiente proyecto será crear un nuevo conjunto de caracteres en RAM que puede utilizarse para escribir en la pantalla, producir listados, etc. de hecho, será idéntico en todos los aspectos menos en la apariencia. El manu del Spectrum sugiere que esto se puede hacer, pero ofrece muy poca inform ción para ello. El nuevo conjunto de caracteres será colocado sobre la RAM TOP, debajo de los caracteres para gráficos definidos por el usuario. Se facilitará una guía paso por paso de cómo hacer esto y como redefinir cualqui carácter que desee. Todo depende de que las variables 23606/7 del sistem apunten al comienzo del conjunto de caracteres. Las direcciones, donde se posible, se encajan con los Spectrums de 16K de RAM y de 48K de RAM menos que haya un método para calcular las direcciones para máquinas co cualquier cantidad de memoria. El nuevo conjunto de caracteres contendrá u texto de estilo itálico con esta apariencia:

```
Esta es una impresion que mues-
tra EL NUEVO CONJUNTO DE CARAC-
TERES en accion. Solo se han
redefinido las letras y los nume
ros 0123456789, aunque mas tarde
veremos como redefinir cualquier
caracter deseado.
```

```
De momento, los simbolos tales
como $#£!% permanecen sin cambios
aunque pueden ser redefinidos
si se desea. A continuacion
Mostramos un listado producido
con este conjunto de caracteres
```

```
Este listado convierte al Spec-
trum en una maquina de escribir
sencilla. Puede borrar el ultimo
caracter pulsado, usando DELETE
(Caps-Shift/0). Pulse enter para
iniciar una nueva linea de texto
```

```
  5 BORDER 0
 10 PRINT INKEY$;
 20 IF INKEY$<>"" THEN GO TO 20
 30 IF INKEY$="" THEN GO TO 30
 34 BEEP .01,25
 37 IF INKEY$=CHR$ 12 THEN PRIN
T CHR$ 8;" ";CHR$ 8;: GO TO 20
 40 GO TO 10
```

Paso 1 El primer paso es bajar la dirección de la RAMTOP en 768 bytes necesarios (por el generador de caracteres) para hacer espacio entre el final del BASIC y los gráficos definidos por el usuario en los cuales se almacenará el nuevo grupo de caracteres. Como en la versión de la ROM, en el nuevo conjunto de caracteres hay 768 bytes.

Spectrums de 16K: La nueva RAMTOP que se necesita será de 31831, en vez del valor normal de 32599. Para determinar la RAMTOP al nuevo valor use el comando CLEAR 31831.

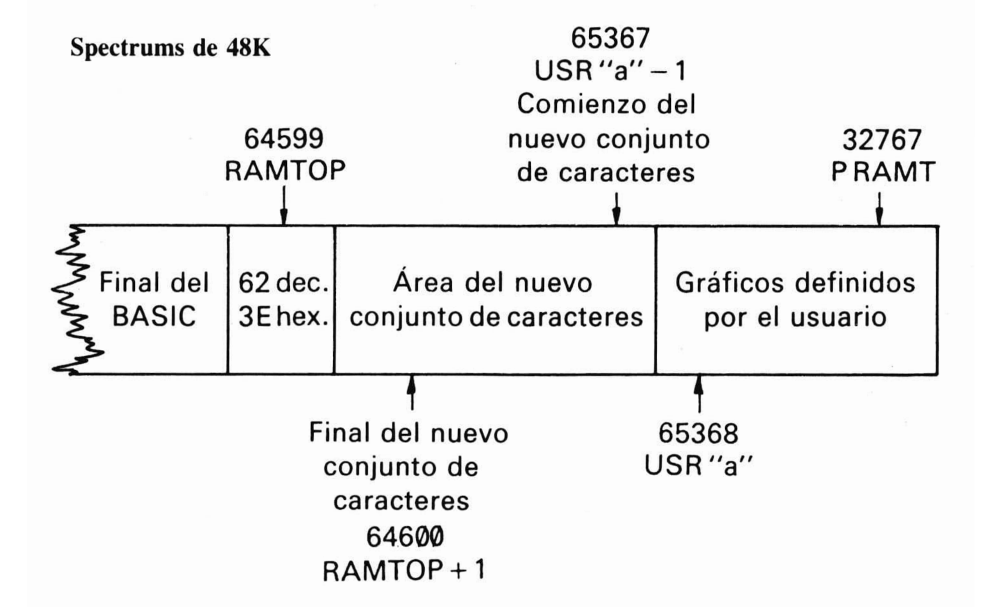
Spectrums de 48K: La nueva RAMTOP que se necesitará será de 64599, al contrario del valor normal de 65367. Para colocar la RAMTOP a este valor entre el comando CLEAR 64599.

Estas dos versiones del comando se basan en números absolutos. Si Vd. posee una cantidad diferente de memoria conectada, necesitará reemplazar los números con una expresión que permita que los valores correctos se correspondan con las circunstancias. Esto también sucedería en el caso de que Vd. tuviera rutinas en código de máquina almacenadas sobre el RAMTOP. Esta expresión desvaloraría la RAMTOP de su actual dirección por 768 bytes cada vez que se utilizara para buscar la dirección de la RAMTOP en la variable del sistema 23730/1, restando 768 de esto y usando el CLEAR con este valor de la siguiente forma:

```
CLEAR (PEEK 23730+256*PEEK 2373
1-768)
```

Si las direcciones necesarias para las máquinas de 48K no aparecen en las siguientes páginas, no importa, en la mayoría de los casos se convierten fácilmente sumando 32768 a los valores de las máquinas de 16K, excepto, naturalmente, para las variables del sistema. A continuación mostramos un diagrama de cómo queda el mapa de memoria:

Spectrums de 16K



Paso 2 Ahora el conjunto de caracteres existente debe copiarse de la ROM a esta área de la RAM de manera que los que no se quieran reprogramar permanecerán igual a como estaban siempre.

Versión del Spectrum de 16K:

```
10 FOR a=15616 TO 16383
20 POKE 16216+a,PEEK a
30 NEXT a
```

Versión del Spectrum de 48K:

```
  1 REM Copiador 16K

10 FOR a=15616 TO 16383
20 POKE 48984+a,PEEK a
30 NEXT a
```

Es importante que esto se entre correctamente ya que cualquier error sería muy difícil de corregirse más adelante y Vd. debería empezarlo todo otra vez.

Paso 3 Ahora vamos a comenzar con la redefinición. Primero le diré cómo redefinir al estilo itálico los números, luego las letras mayúsculas y, finalmente, las minúsculas.

En primer lugar, pasemos a redefinir los números. Entre este programa:

```
 1 REM Cargador 16K

10 FOR a=1 TO 80
20 INPUT B
30 POKE 31959+a,B
40 NEXT a
```

```
 1 REM Cargador 48K

10 FOR a=1 TO 80
20 INPUT B
30 POKE 64727+a,B
40 NEXT a
```

Entre estos números como respuesta a las sentencias INPUT durante la ejecución del programa de arriba. Primero entre la primera hilera, de izquierda a derecha, después la segunda hilera y así sucesivamente...

DATOS PARA LOS NUMEROS

0	60	70	74	148	164	120	0
0	48	80	16	32	32	248	0
0	28	34	4	56	64	124	0
0	30	4	24	4	72	56	0
0	6	10	20	36	126	8	0
0	30	16	60	2	68	56	0
0	30	32	60	66	68	56	0
0	30	2	4	8	16	32	0
0	28	36	56	68	68	56	0
0	28	34	34	28	4	56	0

Observe como Vd. todavía no verá ningún efecto, más adelante cambiaremos una determinada variable del sistema.

Paso 4 Ahora redefiniremos las letras mayúsculas. Use este programa para entrar los ¡208 números que siguen!.

```
16K CARGADOR MAYUSCULAS

  10 FOR A=1 TO 208
  20 INPUT B
  30 POKE 32095+A,B
  40 NEXT A
```

```
48K CARGADOR MAYUSCULAS
  10 FOR A=1 TO 208
  20 INPUT B
  30 POKE 64863+A,B
  40 NEXT A
```

DATOS PARA LAS MAYUSCULAS

0	12	16	34	62	66	66	0
0	28	18	60	34	66	124	0
0	28	34	32	64	68	56	0
0	24	20	34	34	68	120	0
0	30	16	60	32	64	120	0
0	30	16	60	32	64	64	0
0	28	34	32	76	68	56	0
0	18	18	60	36	72	72	0
0	62	8	16	16	32	248	0
0	2	2	4	68	72	56	0
0	18	20	56	40	68	66	0
0	16	16	32	32	64	124	0
0	34	54	42	66	68	68	0
0	18	26	42	44	68	68	0
0	28	34	34	68	68	56	0
0	28	18	34	60	64	64	0
0	60	66	66	164	148	120	0
0	28	18	34	60	68	66	0
0	28	34	24	4	68	56	0
0	62	8	8	16	16	32	0
0	17	34	34	68	68	56	0
0	34	34	36	36	40	16	0
0	33	33	66	66	90	36	0
0	34	20	24	56	68	130	0
0	34	20	8	16	32	64	0
0	62	4	8	16	32	124	0

Paso 5 Ahora redefiniremos las letras minúsculas. Use este programa para entrar los siguientes números:

```
16K CARGADOR MINUSCULAS

  10 FOR A=1 TO 208
  20 INPUT B
  30 POKE 32351+A,B
  40 NEXT A

48K CARGADOR MINUSCULAS

  10 FOR A=1 TO 208
  20 INPUT B
  30 POKE 65119+A,B
  40 NEXT A
```

DATOS PARA LAS MINUSCULAS

0	0	12	2	60	68	56	0
0	16	16	60	34	66	124	0
0	0	28	32	32	64	56	0
0	2	2	28	36	68	56	0
0	0	28	34	124	64	56	0
0	6	8	12	16	16	32	0
0	0	14	18	34	60	4	120
0	16	16	62	34	68	68	0
0	4	0	24	8	16	120	0
0	2	0	4	4	8	72	48
0	16	20	56	48	72	68	0
0	8	16	16	32	32	24	0
0	0	54	73	73	146	146	0
0	0	60	34	34	68	68	0
0	0	28	34	36	68	56	0
0	0	28	18	34	60	64	64
0	0	30	18	36	60	8	30
0	0	14	16	16	32	32	0
0	0	30	32	24	4	120	0
0	4	30	8	16	18	12	0
0	0	34	34	68	68	56	0
0	0	34	34	36	40	16	0
0	0	65	65	146	146	108	0
0	0	34	20	24	40	68	0
0	0	18	36	60	8	16	96
0	0	60	8	16	32	120	0

Paso 6 Usándolo. Supongo se sentirá contento de saber que hemos terminado de escribir. Para usar este nuevo conjunto de caracteres tenemos que alterar el valor CHARS de la variable del sistema 23606/7 (vea el manual del Spectrum c.25) que es el puntero del comienzo del conjunto de caracteres. Generalmente tiene un valor de 256 menos que el comienzo del primer byte del generador de caracteres para facilitar el encuentro de la dirección de los datos para cada carácter. ¿Cómo?. Simplemente permite una simple manipulación matemática de los CODIGOS, significando que la dirección de comienzo de caracteres

individuales puede ser hallada desde el valor contenido en 23606/7 y el CODIGO de estos caracteres multiplicado por 8; normalmente el 23606/7 tiene los valores 0 y 60 respectivamente cuando se usa el conjunto de caracteres de la ROM en ambos Spectrums, el de 16K y el de 48K, o sea:

PEEK23606 es 0 } 0 + 256 * 60 es 15360
PEEK23607 es 60 }

Para hacer este apunte al nuevo conjunto de caracteres, en un Spectrum de 16K usaríamos:

POKE23606,88:POKE23607,123 (88+256*123=31576)

En un Spectrum de 48K sería:

POKE23606,88:POKE23607,251 (88+256*251=64344)

```
VERSION 16K

POKE 23606,88: POKE 23607,123

VERSION 48K

POKE 23606,88: POKE 23607,251
```

Es mejor entrar ambos como un comando directo largo unido por dos puntos en la forma normal,pues si se entran como comandos individuales, la instrucción se ejecutará de mitad en mitad y a la primera mitad, aparecerá la pantalla llena de signos incomprensibles (se hará muy difícil comprobar que el segundo POKE se está entrando correctamente). Si todo lo ha hecho bien, lo que escriba ahora aparece con un nuevo tipo de letra, aunque nada de lo que había en la pantalla permanece como estaba. Los valores POKEados dieron a 23606/7 un valor de 31576 para los usuarios de 16K y 64344 para los usuarios de 48K. Para volver al conjunto de caracteres de la ROM en ambas máquinas de 16K y 48K use:

POKE23606,0: POKE23607,60

Vd. puede utilizar ambas como sentencias de programas de manera que Vd. pueda volver a cambiarlas y así sucesivamente entre ambos conjuntos de caracteres durante un programa si Vd. quiere. Como todo lo que se ha impreso permanece en la pantalla a menos que lo sobreescriba o borre, puede libremente mezclar varios tipos de letras en la pantalla si Vd. desea. También puede

hacer listados de la impresora en caracteres escalonados si desea. El conjunto de caracteres escalonados tiene una gran ventaja con una impresora ZX ya que usando el conjunto vertical normal muestra cualquier deficiencia en la calidad de la impresión. En esto ayuda el nuevo conjunto de caracteres escalonados. En realidad, la apariencia es lo único que le induce a un problema. El SCREEN$ identifica los caracteres en la pantalla buscando en el generador de caracteres un carácter de acoplo desde el cual puede decirle qué carácter es. Si Vd. está usando el nuevo conjunto de caracteres pregunte SCREEN$ para identificar un carácter impreso con el antiguo conjunto de caracteres de la ROM, no será identificado y retorna la cadena vacía. El SCREEN$ debe buscar en el mismo conjunto de caracteres desde el cual fue originalmente escrito el carácter en la pantalla. ¡Esto es sólo un problema si es que Vd. está cambiando entre dos o más conjuntos de caracteres!.

Paso 7 Grabación y lectura en la cinta. Antes de hacer nada más, grave el nuevo conjunto de caracteres en la cinta de forma que Vd. pueda cargarlo al computador cuando Vd. lo necesite otra vez. Me complace decir que es más fácil cargar y usar que entrar todo de nuevo.

Para grabar el nuevo conjunto de caracteres en la cinta desde un Spectrum de 16K:

SAVE"letras"CODE 31832,768

Para grabar el nuevo conjunto de caracteres en la cinta desde un Spectrum de 48K:

SAVE"letras"CODE 64600,768

Para volver a cargar el nuevo conjunto de caracteres desde la cinta a su Spectrum de 16K, escriba y entre:

CLEAR 31831:LOAD"letras"CODE 31832,768

En una máquina de 48K:

CLEAR 64599: LOAD"letras"CODE64600,768

Recuerde VERIFICAR después de salvar el conjunto de caracteres porque si algo ha ido mal tendrá que reescribirlo todo otra vez. Incidentalmente, LOAD""CODE recargará otra vez el conjunto de caracteres en el mismo lugar en que se grabó, mejor salvar un poco cuando se escribe. Así es como aparecerá el nuevo conjunto de caracteres:

Conjunto normal de caracteres de la ROM:

```
  ! # $ % & ' ( ) * + , - . / Ø
1 2 3 4 5 6 7 8 9 : ; < = > ? @
A B C D E F G H I J K L M N O P
Q R S T U V X Y Z [ \ ] ↑ _ £ a
b c d e f g h i j k l ▒ n o p q
r s t u v x y z { | } ~ ©
```

Conjunto alternativo de caracteres de la RAM:

```
  ! " # $ % & ' ( ) * + , - . /
Ø 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [ \ ] ↑ _
£ a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~ ©
```

Excepto para las restricciones de la SCREEN$ ambos conjuntos trabajan exactamente igual. Las funciones, palabras del BASIC, etc., funcionan independientemente del tipo de caracteres usado. Solamente las letras y números han sido redefinidos, de manera que, si Vd. desea redefinir caracteres individuales en lugar de un trabajo como el anteriormente descrito, a continuación verá cómo se hace. Vuelva hasta el *Paso 2* de arriba, después entre el programa. Las dos versiones, una para 16K y la otra para 48K:

```
Redefinicion Caracteres (16K)
   5 POKE 23606,88: POKE 23607,1
23
  10 INPUT "Que caracter desea d
efinir ? ";c$
  20 IF c$="" THEN STOP
```

```
 30 IF c$<" " OR c$>"©" THEN GO
TO 10
 40 LET c=CODE c$
 50 FOR a=0 TO 7
 60 INPUT ("¿Que valor para la h
ilera ";a+1;" ? ");valor
 70 POKE 31576+c*8+a,valor
 80 PRINT AT 0,0;c$
 90 NEXT a
100 GO TO 10
```

```
  1 PRINT "Redefinicion Caract
eres (48K)"
```

```
  5 POKE 23606,88: POKE 23607,2
51
 10 INPUT "¿Que caracter desea d
efinir ? ";c$
 20 IF c$="" THEN STOP
 30 IF c$<" " OR c$>"©" THEN GO
TO 10
 40 LET c=CODE c$
 50 FOR a=0 TO 7
 60 INPUT ("¿Que valor para la h
ilera ";a+1;" ? ");valor
 70 POKE 64344+c*8+a,valor
 80 PRINT AT 0,0;c$
 90 NEXT a
100 GO TO 10
```

Una vez haya copiado el conjunto de caracteres desde la ROM dentro de la RAM, ejecute (RUN) el programa. La línea 10 le pregunta qué carácter desea cambiar. Por ejemplo, si Vd. quiere cambiar el símbolo del paréntesis "(" en otro estilo, entre (.

Si quiere parar el programa, pulse ENTER y el programa se detendrá si Vd. pulsa ENTER para entrar una cadena nula. La línea 30 restringe los caracteres que pueden ser redefinidos desde el ESPACIO hasta el símbolo del copyright © (CHR$ 32 a 127).

c es el CODIGO del caracter a redefinir. Se utiliza para determinar dónde está el carácter (su dirección). El bucle en la línea 50 le permite cambiar todos los ocho bytes del patrón de puntos del carácter en la RAM. La línea 60 le pide entrar el nuevo valor para las ocho hileras de los patrones de puntos. Esto

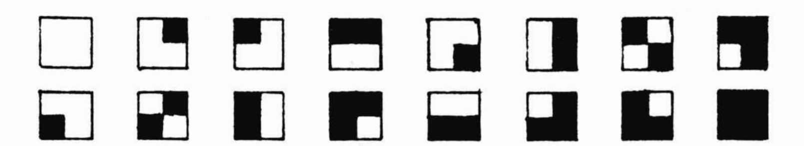
podría hacerse de dos maneras. Debe hacerlo como si se tratara de los GRAFICOS DEFINIDOS POR EL USUARIO ; escribir sus patrones binarios, o sea:

HILERA \ VALOR	128	64	32	16	8	4	2	1	
1	0	1	0	1	0	1	0	1	64 + 16 + 4 + 1 = 85
2	0	0	0	0	0	0	0	0	= 0
3	1	0	0	0	0	1	0	0	128 + 4 = 132
4	0	0	0	0	0	0	0	1	= 1
5	0	0	0	0	0	0	1	1	2 + 1 = 3
6	0	0	0	0	0	0	0	0	= 0
7	0	1	1	1	0	0	0	0	64 + 32 + 16 = 112
8	1	1	1	1	1	1	1	1	128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255

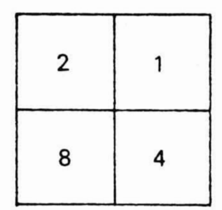
Así pues, por ejemplo, para la hilera 1 podría entrar BIN 01010101, entrar 85 si decide escribir en números decimales. La línea 70 indica donde POKEar este número. La línea 80 se mantiene imprimiendo el nuevo carácter en la parte superior izquierda de la pantalla de manera que Vd. pueda verlo cambiar. La línea 100 envía el programa a la línea 10 para redefinir otro nuevo conjunto de caracteres. Quizás Vd. desea añadir una LINEA en la sentencia INPUT en la línea 10, especialmente si tiene que entrar comillas para redefinir".

GRAFICOS DEL BLOQUE

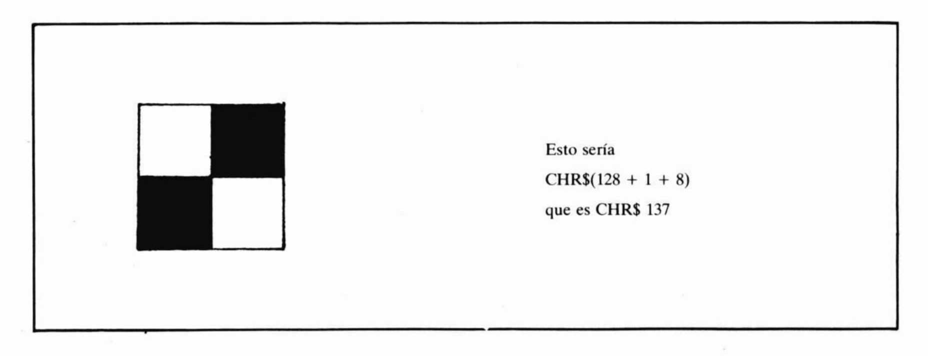
Estos son los símbolos "cuadrados" del CHR$ 128 al CHR$ 143 que aparecen así (ver también el manual del Spectrum el Apéndice A):



A primera vista, no da la sensación de existir un orden especial en los diagramas de arriba. No obstante, Vd. puede entender el porqué de ese orden observando el diagrama de abajo, y teniendo en cuenta los números contenidos en el mismo:



Donde se acoplen cada una de las partes de los cuadrados, añada a 128 el número o los números del diagrama y el total es el CODIGO del carácter que Vd. anotó. Por ejemplo, tome este símbolo:



Otra manera de ver esto es considerar los bits individuales del CODIGO del carácter. Los bits del 0 al 3 del CODIGO son los importantes:

Si el bit 0 se pone a 1, la parte superior derecha del carácter se selecciona. Si el bit 1 se pone a 1, la parte superior izquierda del carácter se selecciona. Si el bit 2 se pone a 1, la parte inferior del cuadrado derecho del carácter se selecciona. Si el bit 3 se pone a 1, se selecciona el cuadrado inferior izquierdo del caracter.

bit1	bit0
bit3	bit2

LIBRERIA DE SUBRUTINAS

Esta sección le facilita una colección de subrutinas, las cuales puede grabar individualmente en cinta para hacer MERGEs dentro de sus programas cuando las necesite. Por supuesto que pueden servirle de base para una serie de subrutinas que Vd. mismo se haga. Todas tienen números diferentes de líneas de forma que puede usar varias dentro de su programa sin que se superpongan. Están numeradas a partir de la línea 9000. Algunas de las subrutinas utilizan los mismos nombres de variables siempre que se usen en las mismas funciones y resulte ventajoso utilizar las mismas variables. La descripción de las subrutinas incluye detalles sobre los nombres de las variables usados y los números de línea, de manera que Vd. sepa lo que cambiar en caso de ser necesario.

1. Borrar una parte de la pantalla

Números de línea usados: 9000 al 9045

Nombres de variables usados: LINEAS,F

Esta rutina borrará el número de líneas en la parte inferior de la pantalla especificado por el programador para los atributos permanentes actuales. La posición del PRINT se mueve a la parte superior izquierda del área borrada, y reinicializa la posición del PLOT a la parte inferior izquierda de la pantalla (coordenadas 0,0) como si fuera después del CLS. Se le pregunta entrar un número del 0 al 22 para decirle a la rutina cuántas líneas hay que borrar desde la parte inferior de la pantalla. Vd. puede querer omitir esta línea INPUT y simplemente especificar las líneas de la variable antes de llamar la subrutina. Si cambia el mensaje, procure guardarlo lo suficientemente corto para no deslizar nada más de la pantalla. La línea 9025 imprime una cadena de 32 espacios en cada línea a borrar y ofrece la posición OVER 0 en el caso de que el OVER 1 tenga un efecto global, en cuyo caso ¡no se borrará nada!. No hay ningún control de color especificado de manera que Vd. puede incluir algo como PAPER 8; BRIGHT 8; FLASH 8; para preservar los atributos del fondo de la pantalla, borre el texto y los gráficos pero deje igual el fondo. La línea 9010 rehúye los INPUTs no válidos haciéndolos reentrar. La línea 9035 mueve la posición PRINT a la parte superior izquierda de la pantalla borrada. La línea 9040 mueve la posición PLOT a la parte inferior de la pantalla haciendo un POKE 0 dentro de las variables del sistema que contienen las actuales coordenadas del PLOT.

```
9000 REM Borrar parte de la pant
alla
9005 INPUT "¿cuantas lineas? ";LI
NEAS
9010 IF LINEAS<0 OR LINEAS>22 OR
```

```
 LINEAS<>INT LINEAS THEN GO TO 9
005
9015 IF LINEAS=0 THEN RETURN
9020 FOR F=21 TO 22-LINEAS STEP
-1
9025 PRINT AT F,0; OVER 0;"
                              "
9030 NEXT F
9035 PRINT AT F+1,0;
9040 POKE 23677,0: POKE 23678,0
9045 RETURN
```

2. Contestaciones Sí o No

Números de línea usados: 9050 al 9080

Nombres de variables usados: R$

Esta rutina le permite responder al tipo de preguntas que necesitan una contestación SI o NO. Cualquier palabra que comienza con una n o N se convierte en una N y cualquier palabra que comienza con una s o S se convierte en una S. A partir de aquí, se puede tomar una acción alternativa. Se debe de contestar con sí o no. La rutina no le permitirá continuar salvo que entre una palabra que comience con s, S, n o N. S, n, o N. Al retorno de esta subrutina, R$ contendrá S o N para SI o NO, respectivamente. Las letras en minúsculas se convierten en mayúsculas. La línea 9055 fija la longitud de R$ (la cadena de la contestación) a un carácter. Esto significa que si tan sólo pulsa ENTER, la cadena de la contestación contiene un espacio, el cual no es válido, de manera que pueda volverse a entrar la respuesta. Si se entra una palabra de más de una letra, sólo la primera se pone dentro de R$ debido al dimensionado. Esto confirma en la línea 9075, la cual sólo necesita asegurarse de que R$ es S o N.

```
9050 REM si o no?
9055 DIM R$(1)
9060 INPUT "si o no? ";R$
9065 IF R$="s" THEN LET R$="S"
9070 IF R$="n" THEN LET R$="N"
9075 IF R$<>"S" AND R$<>"N" THEN
 GO TO 9060
9080 RETURN
```

3. Activar CAPS LOCK

Números de línea usados: 9085 a la 9100

Nombres de variables usados: ZZZ

Este programa le fija el CAPS LOCK automáticamente mediante una rutina en código de máquina de seis bytes que puede entrarse desde el teclado sin la necesidad de ninguna otra forma del programa cargador.

Al rastrear el teclado es usualmente deseable que la respuesta sea sólo mayúsculas o sólo minúsculas, normalmente es molesto tener que comprobar si una respuesta válida está en mayúsculas o minúsculas. El problema es que Vd. nunca puede estar seguro de si el usuario va a enclavar el CAPS LOCK o no. Todo lo que la rutina hace es activar el bit 3 de la variable 23568 del sistema a 1 para enclavar el CAPS LOCK. Entonces si el programa mira el teclado de forma que el usuario no puede cambiar el enclave del CAPS (es decir, usando el INKEY$ o IN para buscar en el teclado) no necesitamos preocuparnos más de si la respuesta está en mayúsculas o minúsculas. El código de máquina está contenido en una sentencia REM en la línea 9095 (la cual sólo contendría estos seis caracteres después del REM, ningún control de color, etc., lo que ciertamente confundiría a la pobre máquina).

!	(símbolo de admiración)
j	(j minúscula)
\	(carácter 92. Observe que éste no es el símbolo de la división. Es el caracter en la tecla D obtenido en modo extendido, SHIFT D).
THEN	(la tecla THEN, no las cuatro letras T H E N).
OVER	(la sentencia OVER, no las cuatro letras O V E R).
< >	(símbolo de no igual (o diferente) o el SIMBOLO SHIFT W).

La rutina en código de máquina es llamada con la ayuda de una variable del sistema llamada NXTLIN, la cual nos ayuda a encontrar dónde se encuentra la subrutina en la memoria ya que necesitamos esta información para saber dónde comienza el código de máquina. Realmente la NXTLIN nos da la dirección del comienzo de la siguiente línea del programa tal como su nombre implica. Hay dos bytes para el número de líneas, dos bytes para el marcador de la longitud de la línea y uno para el carácter REM. Esto es porque añadimos el 5 en la línea 9095. La variable ZZZ no es importante para la rutina porque solamente acepta el número que se retorna por la rutina en código de máquina. Puede ser cualquier variable que no se use.

```
9085>REM Activar caps lock
9090 LET ZZZ=USR (PEEK 23637+256
*PEEK 23638+5)
9095 REM !j\ THEN OVER <>
9100 RETURN
```

4. Desactivar el CAPS LOCK

Números de línea usados: 9105 al 9120

Nombres de variables usados: ZZZ

Esto hace lo contrario de la rutina anterior; desenclava el CAPS LOCK para que las letras en minúscula puedan detectarse. Esto, naturalmente, no se tiene en cuenta, cuando se usa el INKEY$ simplemente pulsando la tecla CAPS SHIFT, aunque es un poco menos útil. Excepto para la rutina en código de máquina y los números de línea, la rutina es similar a la anterior. Los seis caracteres después del REM en la línea 9115 son:

!	(símbolo de admiración)
j	(j minúscula)
\	(carácter 92 en modo extendido, SHIFT D *no* el símbolo de dividir).
THEN	(la tecla THEN, no las cuatro letras separadas T H E N).
O˙	(gráficos O, es decir, CHR$ 158).
< >	(no es el signo igual, o el SIMBOLO SHIFT W).

```
9105 REM Desactivar caps lock
9110 LET ZZZ=USR (PEEK 23637+256
*PEEK 23638+5)
9115 REM !j\ THEN <>
9120 RETURN
```

5. Pulse cualquier tecla para continuar

Números de línea usados: 9125 al 9140.

NINGUN nombre de variable utilizado.

A menudo es necesario hacer una espera en el programa hasta que se recibe una señal desde el operador, por ejemplo, mientras se leen las instrucciones. Sin embargo, la mayoría de los programadores incluyen el mensaje "Pulse una tecla para continuar" y sus rutinas no tienen en cuenta la pulsación de las teclas SHIFT, de manera que no es cierto el mensaje ya que no funciona con cualquier tecla. Esta rutina busca en la fila inferior del teclado mediante la función IN para verificar si se han pulsado las teclas CAPS SHIFT y SYMBOL SHIFT. La

rutina puede fallar si se pulsan dos teclas a la vez de las tres filas de la parte superior del teclado, pero esto resulta una combinación harto sorprendente.

```
9125 >REM Pulsar una tecla
9130 PRINT "Pulsa una tecla para
 continuar."
9135 IF INKEY$="" AND IN 65278=2
55 AND IN 32766=255 THEN GO TO 9
135
9140 RETURN
```

6. Uso del SCREEN$ para detectar gráficos definidos por el usuario en la pantalla

Números de línea usados: 9145 al 9195

Nombres de variables usados: X,Y,A,B,A$

Un inconveniente de la función SCREEN$ es que sólo puede detectar los caracteres con los CODIGOS del 32 al 127 y sus reversos en la pantalla. El SCREEN$ funciona recogiendo la dirección del comienzo del generador de caracteres desde las variables del sistema y verificando si hay un acoplo del carácter en la pantalla. Para hacer que el SCREEN$ detecte gráficos, es necesario hacer que el computador piense que el principal conjunto de caracteres comienza en el mismo lugar donde el usuario definió los gráficos para que así estos patrones de puntos puedan ser revisados. Una pequeña manipulación aritmética hará que el SCREEN$ retorne un carácter con el CODIGO correcto. Para usar la rutina Vd. necesita especificar dos variables, la Y y la X. Estas son para SCREEN$ (Y,X), siendo la Y el número de la coordenada Y horizontal en la pantalla, y la X el número de la coordenada X horizontal en la pantalla que Vd. desea examinar. Los valores originales de las variables del sistema que contienen la dirección del actual generador de caracteres están preservados en las variables A y B de manera que pueden ser refijadas después de utilizar la rutina. Esto se hace de manera que Vd. puede usar un conjunto de caracteres que está en la ROM y la rutina refijará la correcta. Si Vd. tiene más de un conjunto de gráficos definidos por el usuario, la rutina sólo revisará primero los caracteres normales, después, si no encuentra coincidencia, revisará los gráficos definidos por el usuario. Si el resultado da un carácter identificado, el CODE de dicho carácter se cambia para prevenir que el gráfico definido por el usuario A se confunda, por ejemplo, con el ESPACIO. Esto se lleva a cabo en la línea 9180.

```
9145 >REM Pantalla
9150 LET A$=SCREEN$ (Y,X)
9155 IF A$<>"" THEN RETURN
9160 LET A=PEEK 23606: LET B=PEE
K 23607
9165 POKE 23606,PEEK 23675
9170 POKE 23607,PEEK 23676-1
```

```
9175 LET A$=SCREEN$ (Y,X)
9180 IF A$<>"" THEN LET A$=CHR$
(CODE A$+112)
9185 POKE 23606,A
9190 POKE 23607,8
9195 RETURN
```

7. Buscar una copia de una cadena dentro de otra cadena

Números de línea usados: 9200 al 9235

Nombres de variables usados: P,B$,C$

Esta rutina le permite buscar en una cadena una copia de otra cadena. Esto puede resultar útil en, digamos, un juego de aventuras donde Vd. quiere detectar ciertas palabras entradas, o en un sistema de introducción donde Vd. quiere buscar datos. Si se encuentra una copia del C$ en el B$ después al retornar de la subrutina, la variable P contendría el número del elemento en donde comienza la copia. Por ejemplo, buscar ENTE en la palabra SORPRENDENTES retornaría 9. Si no se encuentra ninguna copia, entonces la P devolvería el valor 0. B$ es la cadena principal, y nosotros buscamos en B$ una copia de C$. La rutina trabajará con cadenas de cualquier longitud con los límites de memoria disponibles. Sin embargo, las cadenas muy largas tardarán más tiempo en encontrarse.

```
9200>REM Subcadena
9205 LET P=0
9210 IF LEN C$=0 OR LEN B$=0 OR
LEN C$>LEN B$ THEN RETURN
9215 FOR P=1 TO LEN B$-LEN C$+1
9220 IF B$(P TO P+LEN C$-1)=C$ T
HEN RETURN
9225 NEXT P
9230 LET P=0
9235 RETURN
```

8. Convertir los números decimales en una cadena binaria

Números de líneas usados: 9240 al 9280

Nombres de variables usados: DEC,D$,J)

Cuando se trabaja con posiciones de memoria a veces es necesario examinar el estado de los bits individuales, o los patrones de unos y ceros de un número binario. Aparte de eso, esta rutina puede utilizarse para muchas aplicaciones matemáticas donde es necesario convertir entre decimal y binario, y viceversa. Si Vd. tiene un número J, entonces esta rutina lo convertirá en una cadena D$ de ocho o dieciséis dígitos de manera que Vd. pueda imaginar cada dígito como un bit . Así pues, los números desde el 0 al 255 aparecerán como una cadena de ocho caracteres. Desde el 256 al 65535 aparecerán como una cadena de dieciséis

caracteres. Si Vd. no quiere esta característica de "longitud automática" simplemente borre las líneas 9270 y 9275. Antes de llamar a la rutina defina la J como el número que Vd. desea convertir a binario (o sea, LET J = 255), entonces use el GOSUB 9240. El DEC está hecho para ser una copia del J, de manera que el valor pueda cambiarse si se necesita con la rutina, pero guarde el valor de J cuando retorne desde la subrutina. La división repetida por dos nos ayuda a construir el D$ dentro de una cadena, la cual es el equivalente en binario de la J. Si Vd. desea convertir el D$ en un número decimal use VAL ("BIN" + D$).

```
9240> REM Decimal a Binario
9245 LET DEC=J
9250 LET D$=""
9255 LET D$=STR$ (DEC-INT (DEC/2
) *2) +D$
9260 LET DEC=INT (DEC/2)
9265 IF DEC<>0 THEN GO TO 9255
9270 IF LEN D$<8 THEN LET D$="00
000000"( TO 16-LEN D$) +D$
9280 RETURN
```

9. Convertir una cadena binaria en un número decimal

Números de línea usados: 9285 al 9315

Nombres de variables usados: DEC,SUM,E,E$

Para usar la subrutina defina la cadena E$ como una cadena de dígitos binarios con un 1 representando los unos binarios y un 0 representando los ceros binarios. Por ejemplo, LET E$ = "1101". Entonces el GOSUB 9285 y el DEC harán el equivalente decimal de E$. Naturalmente, esto podría hacerse con números binarios usando BIN, pero es ocasionalmente necesario almacenar información como una cadena de forma que los bits individuales puedan ser revisados: Estas rutinas le permiten trabajar en números binarios y decimales y convertir según su necesidad. Es posible usar el BIN para convertir una cadena binaria a un número decimal (usando el VAL) tal como se muestra como segunda alternativa, pero está sujeto a las limitaciones entre el VAL y el BIN.

```
9285>REM Binario a Decimal
9290 LET DEC=0: LET SUM=1
9295 FOR E=LEN E$ TO 1 STEP -1
9300 IF E$(E)="1" THEN LET DEC=D
EC+SUM
9305 IF E$(E)="0" OR E$(E)="1" T
HEN LET SUM=SUM*2
9310 NEXT E
9315 RETURN
```

```
9285>REM Binario a Decmal
9290 LET DEC=VAL ("BIN "+E$)
9295 RETURN
```

Con ambas versiones de los programas Vd. puede incluir espacios entre los dígitos de E$ para facilitar la lectura espaciando los números así: la cadena "1111111111111111" (dieciséis unos) podría entrarse así: "1111 1111 1111 1111" sin afectar para nada los valores porque la rutina salta cualquier carácter que no sea unos o ceros. Usando el BIN Vd. podría separar los dígitos del número binario con espacios pero cualquier otro carácter podría causar un error C, sin ningún sentido en BASIC.

10. Redondear a 2 posiciones decimales

Números de línea usados: 9320 a 9345

Nombres de variables usados: F$,VALOR,LF

Esta rutina redondea la variable VALUE (VALOR) a 2 posiciones decimales y si es necesario añade un cero para completar las posiciones decimales. Esto es muy útil para cálculos que involucran dinero. El valor es devuelto una vez la cadena F$ está preparada para la impresión o decodificación con el VAL si se necesita manejarla como un número. Por ejemplo, si el valor fuera 0.678 la variable F$ de la cadena sería 0.68. Vd. podría añadir el símbolo E o $ si lo necesita durante la impresión. Antes de llamar a la subrutina declare el VALOR en las líneas de LET VALUE = 89.2345 o INPUT "Entre el coste del elemento"; VALOR. Debido a como el Spectrum maneja los números decimales en la memoria, a veces sucede algún fallo en el cual la posición del tercer decimal es 5. 0.005 se tratará como incorrecto, terminando como 0 en lugar de 0.01. Hay una respuesta muy simple, añadiendo 0.5 en la línea 9325; algo así como 0.500001.

```
9320>REM Dos lugares decimales
9325 LET F$=STR$ (INT (VALOR*100
+5.5) /100)
9330 IF F$(1)="." THEN LET F$="0
"+F$
9335 LET LF=LEN F$-LEN STR$ INT
VAL F$
9340 LET F$=F$+(".00" AND LF=0)+
("0" AND LF=2)
9345 RETURN
```

11. Ayudas del teclado para juegos gráficos

Números de línea usados: 9350 al 9380

Nombres de variables usados: X,Y

Las rutinas descritas aquí muestran como todas las secciones del teclado pueden utilizarse para controlar el movimiento de la pantalla en direcciones determinadas. Ya no son necesarias, por ejemplo, más búsquedas nerviosas de las teclas 5 y 8.

En la primera rutina, la mitad izquierda del teclado mueve hacia la izquierda y la mitad derecha del teclado hace lo mismo pero a la derecha. Con una opción de 20 teclas para moverle en una dirección, no debería tener ningún otro problema en encontrar las teclas correctas para el movimiento. Lo único que tiene que tener en cuenta es no pulsar la tecla CAPS SHIFT y el SPACE pues causan un BREAK y detienen el programa.

El control causa un cambio del valor de la variable X. Esto podría utilizarse como la coordenada PRINT o PLOT de un objeto en la pantalla. Depende de Vd. determinar los valores mínimos y máximos que la X puede tomar.

```
9350 REM Movimiento Izquierda o
Derecha
9355 LET X=X+(IN 61438+IN 57342+
IN 49150+IN 32766<>1020)-(IN 634
86+IN 64510+IN 65022+IN 65278<>1
020)
9360 RETURN
```

La segunda rutina suministra el control para el movimiento en ocho direcciones dividiendo el teclado en cuatro secciones; la hilera de arriba para subir, la mitad izquierda de las dos hileras centrales para la izquierda, la mitad derecha de las dos hileras centrales para la derecha y la hilera inferior de teclas para bajar. Pulsando adecuadamente las combinaciones de éstas puede moverse en diagonal. La X es la coordenada de la posición PRINT a través de la pantalla y la Y es la coordenada de la posición PRINT hacia abajo la pantalla. Otra vez Vd. debería fijar los límites de los valores que estas variables pueden tomarle.

```
9365>REM Mover 4 direcciones
9370 LET X=X+(IN 49150+IN 57342<
>510)-(IN 64510+IN 65022<>510)
9375 LET Y=Y+(IN 65278+IN 32766<
>510)-(IN 63486+IN 61438<>510)
9380 RETURN
```

12. Clasificar una cadena

Números de línea usados: 9385 al 9415

Nombres de variables usados: S,T,S$,U$,USADO

Esta rutina le permitirá clasificar una matriz de cadena por orden alfabético en el supuesto de que Vd. sepa cuántas han de clasificarse. USADO es la variable que dice cuántas cadenas hay en la matriz. Así pues, si Vd. había declarado anteriormente DIM S$ (20,20) y lo había asignadoo a 15 cadenas, USADO tendrá un valor de 15 porque no hay nada que le diga clasificar las otras 5. Las dos variables restantes se usan para controlar los dos bucles, así como una cadena temporal que se utiliza durante el cambio del contenido de dos cadenas de la matriz.

```
9385>REM Clasificación de Cadena
s
9390 FOR S=1 TO USADO-1
9395 FOR T=S TO USADO
9400 IF S$(T)<S$(S) THEN LET U$=
S$(S): LET S$(S)=S$(T): LET S$(T
)=U$
9405 NEXT T
9410 NEXT S
9415 RETURN
```

13. Impresión de matrices de cadenas sin espacios directivos

Números de línea usados: 9420 al 9440

Nombres de variables usados: U,W,S$

Esta rutina le permite imprimir cadenas o matrices de cadenas sin el problema de los espacios al final, sobrepasando la pantalla causando un vacío entre dos palabras. Para usar la rutina reemplace la matriz de la cadena con el nombre de la matriz de la cadena que desea imprimir. W indica qué cadena en la matriz hay que imprimir. Así pues, si Vd. tenía DIN S$ (20,20) y quería imprimir S$(8) Vd. diría LET W = 8 antes de llamar a la subrutina.

```
9420>REM Imprimir sin espacios
         al final
9425 FOR U=LEN S$(W) TO 1 STEP -
1
9430 IF S$(W,U)<>" " THEN PRINT
S$(W, TO U): RETURN
9435 NEXT U
9440 RETURN
```

14. Hacer un blip

Números de línea usados: 9445 al 9466

Nombres de variables usados: Z

La rutina simplemente facilita un tono de caída descendente, el cual sirve para cualquier aplicación que requiera un sonido distinto.-Digamos, cuando Vd. ha sido alcanzado por el disparo de un marcianito.

```
9445>REM Hacer un Ruido!
9450 FOR Z=0 TO 20 STEP 2
9455 BEEP .01,69-(Z*2.5)
9460 NEXT Z
9465 RETURN
```

15. Conversión decimal a hexadecimal

Números de línea usados: 9470 al 9505

Nombres de variables usados: DEC,NUMERO, H$,N1

Supongamos que Vd. tiene un número decimal el cual Vd. desea convertir a formato hexadecimal para programación en código de máquina. Esta subrutina convertirá el valor de DEC en una cadena hexadecimal H$ de forma que el decimal 16 sería 10 en hex. No existe límite en el tamaño del número DEC (siempre dentro de los límites aritméticos del Spectrum) mientras sea positivo. Como dos dígitos hex generalmente equivalen a un byte en la memoria, esta rutina añade un dígito cero al final de manera que la cadena hex H$ resultante pueda utilizarse si es necesario con programas cargadores. Esta función se realiza en la línea 9500, de forma que Vd. puede omitir ésta si no necesita esta característica. Las otras variables son irreales utilizadas para realizar funciones aritméticas en el valor del DEC durante la conversión, realmente sin cambiar el valor de DEC se guarda a la salida desde la rutina.

```
9470 REM decimal a hexadecimal
9475 LET dec=numero
9480 LET h$=""
9485 LET n1=INT (dec-(INT (dec/1
6))*16)
9490 LET h$=CHR$ (n1+48+(7 AND n
1>9))+h$
9495 IF INT (dec/16)<>0 THEN LET
 dec=INT (dec/16): GO TO 9485
9500 IF INT (LEN h$/2)*2<>LEN h$
 THEN LET h$="0"+h$
9505 RETURN
```

16. Conversión hexadecimal a decimal

Números de línea usados: 9510 al 9540

Nombres de variables usados: H$,H,VALOR,DEC

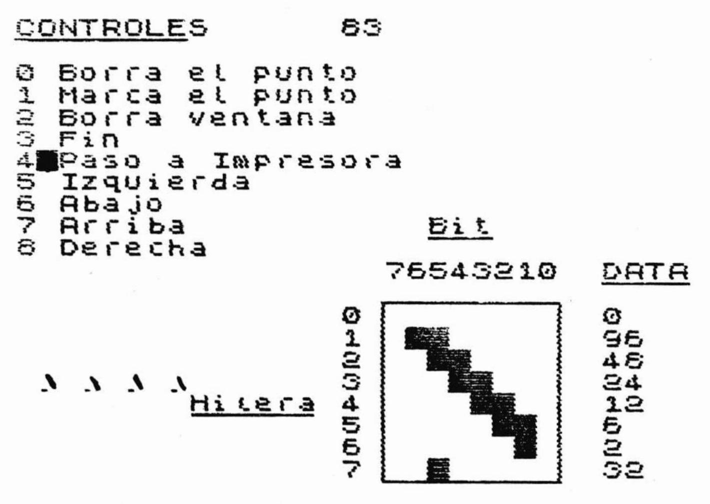
Esta rutina realiza la función contraria, literalmente convierte una cadena H$ hexadecimal a número DEC decimal. El H$ puede tener cualquier longitud tan largo como es el número decimal resultante con las capacidades aritméticas del Spectrum. Se debe definir H$ antes de invocar a la subrutina, el contenido de esta variable queda preservado al salir de la subrutina y el número decimal aparece en la variable DEC.

Los caracteres de la cadena hex deben de consistir en los números del 0 al 9, y las letras mayúsculas de la A a la F o las minúsculas de la f (que se interpretan tanto en mayúsculas como en minúsculas).

```
9510 REM hexadecimal a decimal
9515 LET valor=1: LET dec=0
9520 FOR h=LEN h$ TO 1 STEP -1
9525 LET dec=dec+(CODE h$(h)-48-
(7 AND h$(h)>"9")-(32 AND h$(h)>
="a"))*valor
9530 LET valor=valor*16
9535 NEXT h
9540 RETURN
```

DISEÑO DE GDU (GRAFICOS DEFINIDOS POR EL USUARIO) (SPECTRUM DE 16K)

Este programa le permitirá diseñar los GDUs a su elección y le dirá los valores a usar como datos en un programa para activar dichos gráficos. No incluye los muchos lujos de muchas versiones elaboradas, pero habiendo usado algunas de ellas encuentro que cuando necesito diseñar un caracter para un programa esta simple versión es todo lo que necesito. Todos los comandos y opciones disponibles se muestran en la pantalla todo el tiempo y si Vd. tiene una impresora tiene la opción de tener una copia permanente de la visualización en la pantalla incluyendo un gran tamaño de imagen, igual a lo que es el caracter a tamaño normal y los valores de los datos. Esto es un ejemplo de la visualización de forma que Vd. puede tener una idea de lo que es el programa.



El cuadrado grande es el área de visualización donde aparece el carácter que está diseñando. Estudie esta área usando el programa. El cursor (a + símbolo) muestra el punto actual del gráfico que puede ser sombreado o al contrario. Este cursor puede moverse a la izquierda, hacia abajo, arriba o a la derecha con las teclas del cursor 5, 6, 7 y 8 respectivamente. Pulsando la tecla 0 borrará el punto situado bajo el cursor de forma de que si el cursor + estaba en la parte superior izquierda de la caja, el punto situado en la parte superior izquierda aparecerá en blanco después de pulsar 0 sin tener en cuenta si anteriormente era negro o blanco. Por otra parte, pulsando la tecla 1 convertirá el punto en negro. Observe que el cursor + es blanco sobre puntos negros, de manera que

Vd. siempre puede verlo. Para copias de tamaño "real" de los gráficos definidos por el usuario se imprimen tal cual en la parte izquierda de la versión más reciente de forma que Vd. puede verlo exactamente cómo es.

Pulsando la tecla 2, borra la visualización en la pantalla de los gráficos definidos por el usuario, reinicializa el cursor a la parte superior izquierda de la caja y pone los valores de los datos a 0 cada uno de ellos. Al pulsar la tecla 3, finalizará el programa. Si por cualquier razón quiere recomenzar el programa desde donde se paró (o sea en un error), entonces simplemente pulse CONTINUE. Pulsando la tecla 4, obtendrá una impresión sobre papel (si, naturalmente, tiene la impresora ZX conectada), sin el cursor. Una vez haya terminado con el diseño, copie los valores de los datos de forma que Vd. pueda incorporarlos en una sentencia DATA para los gráficos definidos por el usuario en su programa. Con el ejemplo que se muestra, Vd. escribiría después del número de línea DATA 0,96,48,24,12,6,2,32 como un ejemplo para mostrar en qué orden deberían escribirse los números. Todos los números son decimales. Si Vd. usa el BIN para determinar los gráficos definidos por el usuario, Vd. no usaría este programa ya que los 1s y 0s mostrarían el diseño de los puntos sin la necesidad de diseñarlos con anterioridad. A continuación viene el listado del programa:

```
   1 REM Generador de Graficos
  10 GO SUB 9000
  20 IF INKEY$<>"" THEN GO TO 20
  30 LET I$=INKEY$
  40 IF I$<"0" OR I$>"8" THEN GO
 TO 30
  50 PRINT AT 2+VAL I$,1; INVERS
E 1;">"
  60 IF I$="0" OR I$="1" THEN GO
 SUB 500
  70 IF I$="2" THEN GO SUB 1000
  80 IF I$="3" THEN STOP
  90 IF I$="4" THEN GO SUB 1500
 100 IF I$>="5" AND I$<="8" THEN
 GO SUB 2000
 110 PRINT AT 2+VAL I$,1;" "
 120 GO TO 20
 500 REM Marcar o borrar
 510 LET D$(CY,CX)=I$
 520 LET D(CY)=VAL ("BIN "+D$(CY
))
 530 POKE USR "/"-1+CY,D(CX)
 531 PRINT AT 0,15;cy;cx
 540 PRINT AT 12+CY,16+CX; INVER
SE VAL D$(CY,CX);"+"; INVERSE 0;
 541 PRINT AT 16,1;"/ / / /";AT
12+CY,27;D(CY);"   "( TO 3-LEN ST
R$ D(CY))
```

```
 550 RETURN
1000 REM Borrar pantalla
1010 FOR A=0 TO 7
1020 POKE USR "⁄"+A,0
1030 LET D$(A+1)="00000000"
1040 LET D(A+1)=0
1050 PRINT AT 13+A,17;"          "
;AT 16,1;"⁄ ⁄ ⁄ ⁄";AT 13+A,27;"0
"
1060 NEXT A
1070 LET CY=1: LET CX=1
1080 PRINT AT 12+CY,16+CX;"+"
1090 RETURN
1500 REM Pasar a Impresora
1510 PRINT AT 12+CY,16+CX; OVER
1;"+"
1520 COPY
1530 PRINT AT 12+CY,16+CX; OVER
1;"+"
1540 RETURN
2000 REM Mover Cursor
2010 LET OLDCX=CX: LET OLDCY=CY
2020 LET CX=CX-(I$="5" AND CX>1)
+(I$="8" AND CX<8)
2030 LET CY=CY-(I$="7" AND CY>1)
+(I$="6" AND CY<8)
2040 IF OLDCX<>CX OR OLDCY<>CY T
HEN PRINT AT 12+OLDCY,16+OLDCX;
INVERSE VAL D$(OLDCY,OLDCX);" ";
AT 12+CY,16+CX; INVERSE VAL D$(C
Y,CX);"+"
2050 RETURN
9000 REM Inicio
9010 BRIGHT 0: FLASH 0: INVERSE
0: OVER 0
9020 INK 0: BORDER 7: PAPER 7: C
LS
9030 FOR A=0 TO 7: POKE USR "A"+
A,0: NEXT A
9040 PRINT "CONTROLES"
9041 PRINT
9050 PLOT 0,167: DRAW 63,0
9060 PRINT "0 Borra el punto"
9070 PRINT "1 Marca el punto"
9080 PRINT "2 Borra ventana"
9090 PRINT "3 Fin"
9100 PRINT "4 Paso a Impresora"
9110 PRINT "5 Izquierda"
9120 PRINT "6 Abajo"
9130 PRINT "7 Arriba";TAB 19;"Bi
t": PLOT 152,95: DRAW 23,0
9140 PRINT "8 Derecha"
9150 PRINT TAB 17;"76543210  DAT
A "´´: PLOT 216,79: DRAW 31,0
9160 FOR A=0 TO 7: PRINT TAB 15;
A;TAB 27;"0": NEXT A
9170 PLOT 135,72: DRAW 65,0: DRA
W 0,-65,0: DRAW -65,0: DRAW 0,64
9180 PRINT AT 17,8;"Hilera": PLO
T 65,31: DRAW 44,0
```

```
9190 PRINT AT 13,17;"+"
9200 DIM D(8): DIM D$(8,8)
9210 FOR A=1 TO 8: LET D$(A)="00
000000": NEXT A
9220 LET CY=1: LET CX=1
9230 RETURN
```

Cuando entre este listado, por favor, tenga en mente la siguiente información:

Línea 520: La palabra BIN entre comillas después del VAL es la función BIN, no las tres letras B, I y N.

Línea 530: Entre comillas después del USR, la A se refiere a la A gráfica.

Línea 540: Las letras Aes en "A A A A" son cuatro Aes gráficas.

Línea 1050: Hay 8 espacios en la primera cadena de espacios y 4 Aes gráficas en "A A A A" y el "0" va seguido por dos espacios.

Línea 2040: Un espacio.

Línea 9150: Dos espacios entre los números y el DATA.

Observe los dos apóstrofes después de las comillas.

El programa está diseñado como una serie de subrutinas que se invocan desde un bucle inicial en el programa. Lo primero que hay que hacer es inicializar el programa en la subrutina en la línea 9000. Esto está compuesto de atributos permanentes, etc. (9010 al 9020, texto negro sobre fondo blanco), haciendo que los gráficos diseñados por el usuario en la tecla A tengan todos los puntos borrados para acoplar en la caja en blanco al principio. Esto se hace mediante la sentencia POKE USR 8 veces en un bucle.

Las líneas 9040 a la 9140 imprimen las instrucciones. Las sentencias DRAW son para subrayar. Las líneas 9150 a la 9180 fijan la caja en la cual Vd. edita su carácter. Los números en la parte superior indican a qué bit del punto del byte de DATA corresponden. Los números situados al lado indican los bytes de los datos para los gráficos definidos por el usuario, o sea, la hilera 0 irá dentro del USR "A" + 0, etc.

La línea 9190 imprime el cursor en su posición de comienzo. La línea 9200 determina una matriz numérica D() y una matriz de cadena D$(). El formulario maneja los valores de DATA donde la matriz de cadena maneja una versión binaria para determinar dónde se deben determinar los puntos individuales. Si Vd. quiere imprimir en binario para los usuarios de BIN esta matriz podría imprimirse. La línea 9210 pone el 0 en cada elemento del D$() ya que la visualización comienza con todos los puntos "off". Las variables en la línea 9220 son CY para el cursor la coordenada vertical Y de la pantalla (observe, que no va desde la parte superior de la pantalla) y CX para la X en horizontal, otra vez valores de X no estándar. Retornando desde la subrutina el programa entre el bucle principal en las líneas de la 20 a la 120. La línea 50 imprime un cursor más

grande que un símbolo después de la opción escogida. Esto es opcional y puede omitirse (en cuyo caso también omite la línea 110,la cual borra el marcador una vez que ha terminado la opción,la cual en la mayoría de los casos sólo tarda una fracción de un segundo). Las líneas 60 a la 100 seleccionan la subrutina que hay que ejecutar. Si añadimos una de sus propias subrutinas o cambiamos este programa,tenga cuidado,ya que se toman referencias directas para el carácter I$ en otras partes del programa; en particular, el VAL se aplica y el valor usado determina donde algo está en inverso o no.

La subrutina de la línea 500 se invoca cuando Vd. pulsa la tecla 0 o la 1. Actúa ennegreciendo o blanqueando un punto. Primeramente, el elemento apropiado del D$() es 0 cuando se blanquea o un 1 cuando se ennegrece. Esto se convierte a decimal en la línea 520 aplicando el VAL a una cadena que contiene el BIN y una cadena binaria desde el D$(), el resultado se almacena en el elemento apropiado de una matriz D() con los datos. El D() se pone al día cada vez que se hace un cambio en el GDU. Este valor se POKEa dentro del área de gráficos usados por el usuario en la memoria de forma que la A gráfica se convierte en una copia de la figura que Vd. está editando. La línea 540 imprime el cursor + en la caja de la pantalla y determina donde ha de ser inverso o no desde el carácter en el I$ que Vd. pulsó desde el teclado. También pone al día los valores de los datos en la pantalla imprimiendo el elemento apropiado del D() en la columna a la derecha de la caja de visualización. Observe el uso del LEN STR$ para decidir cuántos espacios hay que imprimir para sobreescribir cualquier valor que estuviera allí anteriormente. Imagine que el DATA de una hilera fuera 127 (o sea, sólo el bit 7 fue enblanquecido), entonces Vd. convirtió en blanco el bit 6, que hizo DATA.63 Esto sólo tendría 2 dígitos en lugar de los tres anteriores,de forma que no lo sobreescribe completamente y deja el 7 en la pantalla dando 637, ¡voilà!. La última sentencia PRINT en la línea 540 decide cuántos espacios debe imprimir, 2, 1, o ninguno dependiendo de cuantos caracteres tenga el número. Siempre se imprimen tres caracteres. La línea 540 también imprime 4 Aes gráficas.

La siguiente subrutina en la línea 1000 borra la visualización de los gráficos que Vd. determinó y reinicializa las matrices de datos para comenzar los valores igual que el cursor y el GDU de A.

La rutina en la línea 1500 copia el contenido de la pantalla en la impresora ZX. Primero se borra el cursor + usando PRINT OVER 1 que, desde que el cursor está en inverso en relación con el fondo de la pantalla, convertirá el cursor igual que el fondo, así pues,en realidad se borra. El COPY (COPIAR) se usa cuando el cursor retorna usando PRINT OVER 1 otra vez. Esta es una aplicación muy útil del OVER 1, borrando y repitiendo algo de la pantalla alternativamente simplemente imprimiendo lo mismo OVER y OVER otra vez. La subrutina en la línea 2000 es una rutina que sirve para mover el cursor usando las teclas del cursor de la 5 a la 8. Observe que la línea 2040,como la cadena binaria D$,se usa

otra vez con el VAL para determinar donde se imprime algo en inverso o no. Inusual, pero ideal para esta aplicación.

También puede observar que no hay repetición en las teclas, o sea, si Vd. quiere moverse desde un lado de la caja al otro, tiene que pulsar la tecla 8 cada vez que quiera moverse un espacio. Esto es debido a la línea 20 que si encuentra que se pulsa una tecla espera a que se deje de pulsar. Como el programa funciona tan rápido, el teclado llega a ser difícil de usar porque el cursor va muy rápido. Si Vd. quiere, pruébelo usando una espera en la línea 20, algo así como 20 FOR A = 1 TO 50: NEXT, con lo cual obtendrá una espera lenta constante. No use el PAUSE (PAUSA) ya que éste detendrá el programa después de cada presión de tecla y Vd. tendrá que seguir la repetición periódicamente y así sucesivamente.

GRAFICOS DEFINIDOS POR EL USUARIO YA PREPARADOS

Hay símbolos de gráficos ya prediseñados para que Vd. los use en sus propios programas de gráficos, de manera que no pierda el tiempo diseñándolos. Consisten en gráficos del usuario definidos más comúnmente usados, con los datos relevantes para su creación colocados en las sentencias DATA para la lectura subsiguiente y POKEarlos dentro del área de memoria de gráficos definidos por el usuario. Aunque la numeración de líneas está por orden aleatorio, Vd. la cambiará para que se acople a sus programas, los listados del DATA se muestran tal como aparecerían en la pantalla en un listado de programa para que Vd. pueda hacer la entrada de correcciones necesaria.

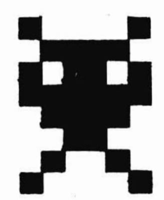
Por ejemplo, para determinar el carácter del invasor:

```
8000 REM Cargar gráfico en "A"
8020 RESTORE
8030 FOR a=USR "a" TO USR "a"+7
8040 READ udg
8050 POKE a,udg
8060 NEXT a
8080 REM Listado de datos
extraídos de otros ejemplos y
acoplados a este programa
8100 DATA 66,60,90,126,60,24,36,
66
8110 PRINT "A": REM Gráfico A
```

Naturalmente, Vd. dejaría las sentencias REM para simplificación. Y esto es todo: ¡hágalo y pruébelos!.

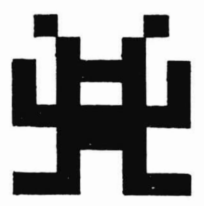
(1) Caracteres del invasor

```
invasor 1
```



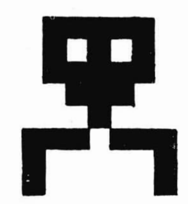
```
8010 DATA 66,60,126,60,24,36,66
```

```
invasor 2
```



```
8030 DATA 66,36,189,165,255,60,3
6,231
```

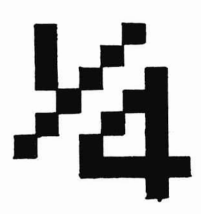
```
invasor 3
```



```
8050 DATA 62,42,62,28,8,119,65,6
5
```

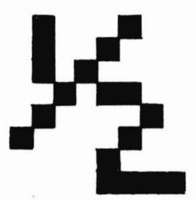
(2) fracciones

```
un cuarto
```



```
8290 DATA 68,72,82,38,74,146,31,
2
```

```
mitad
```



```
8280 DATA 68,72,80,44,66,132,8,1
5
```

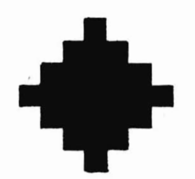
```
tres cuartos
```



```
8310 DATA 225,34,68,42,246,42,79
,130
```

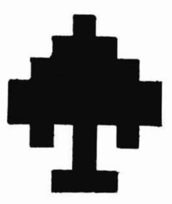
(3) cartas

```
cartas carro
```



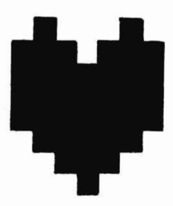
```
8150 DATA 8,28,62,127,62,28.8.0
```

cartas pique



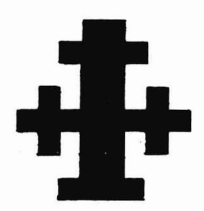
```
8170 DATA 16,56,124,254.254.84.1
6,56
```

cartas corazones



```
8190 DATA 68,238.254,254,254,124
.56,16
```

cartas trebol



```
8210 DATA 24,60,24.90,255.90.24,
60
```

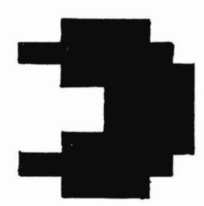
(4) símbolos de comecocos

comecocos mirando a la derecha



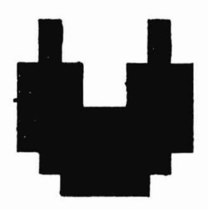
```
8010 DATA 60,127,252,240,240,252
,127,60
```

comecocos mirando a la izquierda



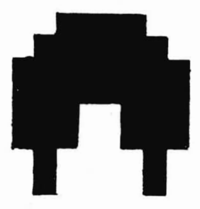
```
8050 DATA 60,254,63,15,15,63,254
,60
```

comecocos mirando hacia arriba



```
8030 DATA 66,66,231,231,255,255,
126,60
```

comecocos mirando hacia abajo



```
8070 DATA 60,126,255,255,231,231
,66,66
```

fantasma



```
8090 DATA 56,124,214,214,254,254
,170,170
```

tableta de energia



```
8110 DATA 0,24,60,126,126,60,24,
0
```

```
punto de comida
```

```
8130 DATA 0,0,0,24.24,0,0.0
```

(5) coches

```
coche mirando a la derecha
```



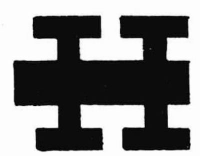
```
8010 DATA 0,238,68,255,255,68,23
8,0
```

```
coche mirando hacia abajo
```



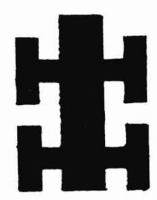
```
8070 DATA 90,126,90,24,90,126,90
,24
```

coche mirando hacia la izquierda



```
8030 DATA 0,119,34,255,255,34,11
9,0
```

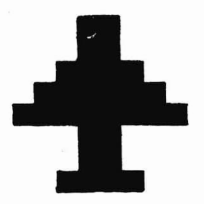
coche mirando hacia arriba



```
8050 DATA 24,90,126,90,24,90,126
,90
```

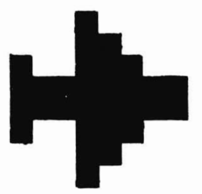
(6) avión

avion 1



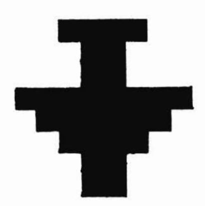
```
8000 DATA 24,24,60,126,255,24,24
,60
```

avion 2



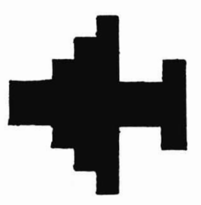
```
8030 DATA 16,24,156,255,255,156,
24,16
```

avion 3



```
8050 DATA 60,24,24,255,126,60,24
,24
```

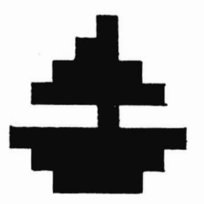
avion 4



```
8070 DATA 8,24,57,255,255,57,24,
8
```

(7) barco

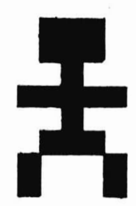
```
barco
```



```
8090 DATA 8,24,60,126,8,255,126,
60
```

(8) hombre

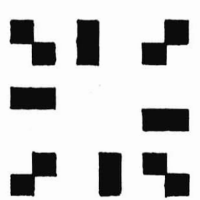
```
hombre
```



```
8010 DATA 56,56,16,124,16,56,68,
68
```

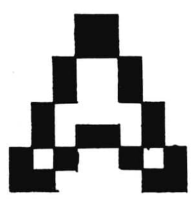
(9) explosión

```
explosion
```

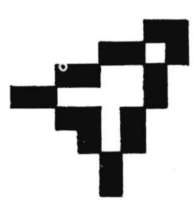


```
6330 DATA 145,82,0,192,3,0,74,13
7
```

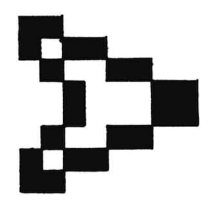
(10) naves en ocho direcciones de usos generales, para juegos espaciales, etc.



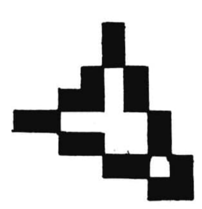
```
8120 DATA 24,24,36,36,66,90,165,
195
```



```
8140 DATA 3,13,50,194,52,20,8,8
```



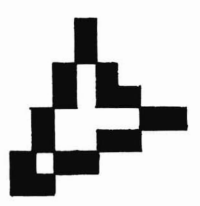
```
8160 DATA 192,176,76,35,35,76,17
6,192
```



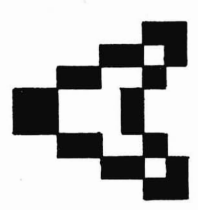
```
8180 DATA 8,8,20,52,194,50,13,3
```



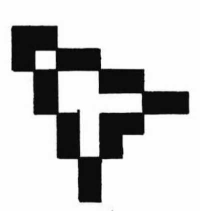
```
8200 DATA 195,165,90,66,36,36,24
,24
```



```
8220 DATA 16,16,40,44,67,76,176,
192
```



```
8240 DATA 3,13,50,196,196,50,13,
3
```



```
8260 DATA 192,176,76,67,44,40,16
,16
```

CLASIFICAR EL SCREEN$ Y EL ATTR

Un requisito frecuente durante la programación de juegos es revisar el carácter en una posición determinada en la pantalla, por ejemplo, detectar colisiones. El BASIC del Spectrum tiene la función SCREEN$ para esta finalidad y funciona así.

La sintaxis es SCREEN$(y,x) la cual retorna una cadena correspondiente al carácter de la pantalla en la hilera y vertical a la pantalla y en la columna x a través de la misma independientemente de si el carácter es inverso o no. La y, y la x pueden ser números o expresiones numéricas como en una sentencia PRINT. Por ejemplo, pruebe este programa:

```
 10 PRINT AT 0,5;"+"
 20 PRINT "El caracter at 0,5 e
s ";SCREEN$ (0,5)
```

Puede verse que el inverso no tiene nada que ver con este programa. El SCREEN$ retornará + para un inverso + en la pantalla o un SPACE (ESPACIO) para un carácter 8 GRAPHICS SHIFT, por ejemplo.

```
 10 PRINT AT 0,5; INVERSE 1;"+"
 20 PRINT "El caracter at 0,5 e
s ";SCREEN$ (0,5)
```

Los atributos (colores/destello/brillo) no afectan al SCREEN$, sólo los puntos que se encuentran actualmente en la pantalla. Aquí hay un ejemplo para demostrar el uso del SCREEN$ en juegos gráficos. Vd. tiene el control de una nave espacial (inverso V) atravesando el espacio tratando de evitar colisionar con las estrellas (asteriscos). Cuando Vd. colisiona, el juego se detiene y aparece su puntuación. La tecla 5 le mueve a la izquierda de la pantalla y la tecla 8 le mueve hacia la derecha. Cuando la acción sucede Vd. se encuentra en la mitad inferior de la pantalla.

La línea 30 determina cuan lejos dentro de la pantalla comienza su nave, la línea 40 incrementa su puntuación y la 50 mantiene el deslizamiento de la pantalla sin que aparezca el mensaje SCROLL de deslizamiento. La línea 60 imprime tres estrellas en la parte interior de la pantalla, entonces borra la posición actual de la nave para el deslizamiento. La línea 70 está atenta al teclado y determina donde se imprime la nave espacial. La línea 80 busca la nueva

posición del carácter usando SCREEN$. Si este carácter no es un asterisco entonces el juego continúa imprimiendo la nave espacial regresando a la línea 40. Si ha habido una colisión se imprime su puntuación y el juego termina con su nave destellando para indicar una colisión.

```
TU PUNTUACION 111* *    *
*          *         *
  *               *    *
     *       *     *
  **             *
 *         *            *
    *            *
 *     *     *
       *   *            *
               *  *        *
         V*          *
  *                *      *
     *      *   *
    *  *             *
       *    *     *
*                    **
            *     *  *
       *          *       *
    *            *        *
  ** *
      *       *       *
```

```
  1 REM Estrellas
 10 RANDOMIZE
 20 LET SCORE=0
 30 LET ATRAVES=INT (RND*32)
 40 LET SCORE=SCORE+1
 50 POKE 23692,255: REM DESLIZA
R
 60 PRINT AT 21,RND*31;"*";AT 2
1,RND*31;"*";AT 21,RND*31;"*";AT
 10,ATRAVES;" ";AT 21,31
 70 LET ATRAVES=ATRAVES-(INKEY$
="5" AND ATRAVES>0)+(INKEY$="8"
AND ATRAVES<31)
 80 IF SCREEN$ (10,ATRAVES)<>"*
" THEN PRINT AT 10,ATRAVES; INVE
RSE 1;"V": GO TO 40
 90 PRINT AT 0,0;"TU PUNTUACION
 ";SCORE;AT 10,ATRAVES; FLASH 1;
"V"
```

Lo que no se le ha dicho es que el SCREEN$ está muy limitado en lo que puede reconocer. Sólo revisa aquellos caracteres en el conjunto de caracteres que van desde el SPACE al símbolo ©, o CHR$(32) al CHR$(127). Los gráficos definidos por el usuario y los gráficos del bloque en las teclas numéricas del teclado no son reconocidos, y es una pena ya que estos caracteres son básicamente usados en programas gráficos donde el SCREEN$ es de mayor uso.

Una forma de resolver este problema es usando ATTR, e imprimir en colores diferentes. Así pues, si Vd. tenía un juego donde disparaba misiles contra un objeto podía hacer que éste fuera verde, el misil rojo y el lanzador de misiles azul, y así sucesivamente. Para verificar si el misil había tocado el objeto debería observar si había verde en el cuadrado del carácter hacia donde se dirigía el misil, con ATTR. Generalmente esto se hace donde el SCREEN$ no puede utilizarse por ninguna razón. Por ejemplo, si el objeto era azul sobre un fondo amarillo de brillo normal y no hacía destello, Vd. podría ver la presencia del objeto así:

```
IF ATTR(Y,X)=49 THEN...
```

teniendo en cuenta que el ATTR retorna un número que es (destello * 128) + (brillo * 64) + (color PAPER * 8) + (color INK).

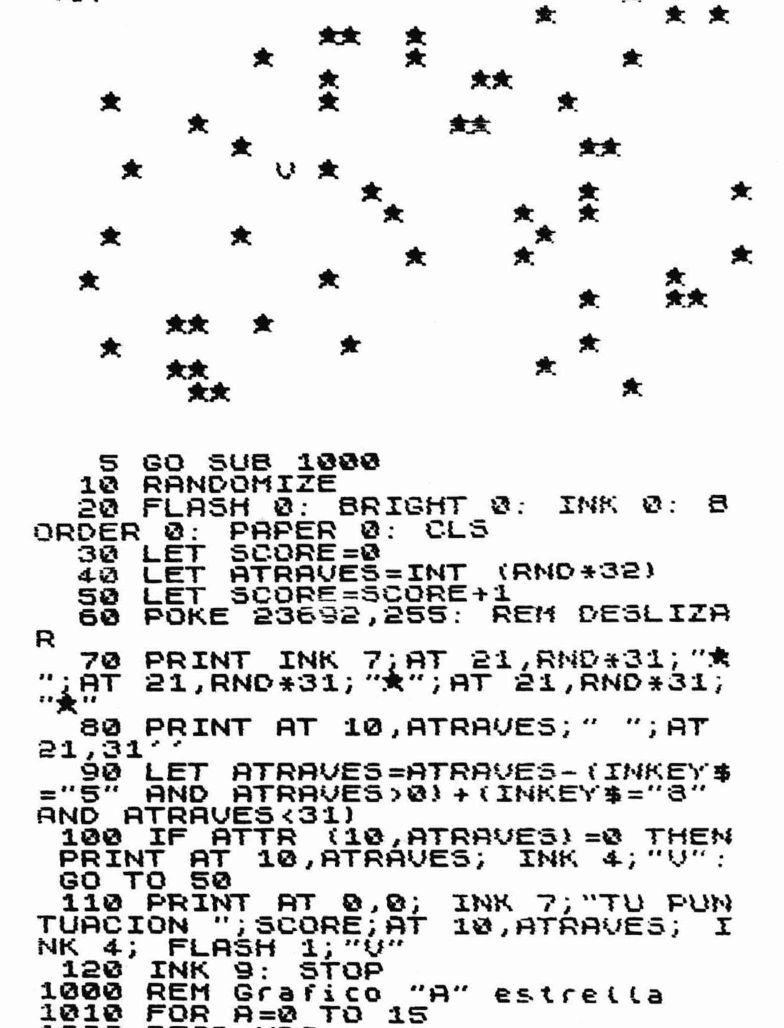
Aquí debería tener en cuenta que si hay algún espacio en blanco en la pantalla debería tener un atributo diferente en relación al resto de gráficos en la pantalla. Si el color global era INK (TINTA) azul y los misiles y espacios estaban ambos impresos en los colores globales entonces el ATTR no puede distinguir entre los espacios y los misiles. Para ello debería especificar un color INK global lo mismo que el color PAPER del fondo antes de borrar la pantalla:

```
FLASH 0:BRIGHT 0:INK 6:BORDER 6:PAPER 6:CLS.
```

Esto determina los atributos de cualquier punto en blanco en la pantalla a 54 (la pantalla aparecerá en color amarillo). Vd. no podrá usar el INK amarillo sobre PAPER amarillo durante el curso del programa para imprimir gráficos porque en la pantalla se imprimirá en colores temporales mientras que todos los blancos se harán con colores permanentes. El único problema es que cuando el programa se detenga Vd. no podrá ver el listado a menos que especifique algo así como INK 9:STOP cuando el programa llegue al final.

A continuación viene el desarrollo del programa de las estrellas que hace uso de los gráficos definidos por el usuario y del ATTR para revisar en la pantalla las colisiones descritas arriba. Los gráficos son tres Aes gráficas en la línea 70 y la B gráfica en las líneas 100 a la 110. La línea 20 determina los atributos globales a 0, de forma que un área de la pantalla sin nada impreso en ella tiene un atributo de 0. Las estrellas están impresas en blanco contra espacios negros para darles un atributo de algo que no sea 0. O sea, cuando el programa busca si una colisión ha sucedido en la línea 100 sólo los espacios vacíos permiten que la acción continúe. Si no hay ningún espacio vacío, ha sucedido una colisión. Aquí está el listado del programa.

Resultado y listado del programa Estrellas 2




```
   5 GO SUB 1000
  10 RANDOMIZE
  20 FLASH 0: BRIGHT 0: INK 0: B
ORDER 0: PAPER 0: CLS
  30 LET SCORE=0
  40 LET ATRAVES=INT (RND*32)
  50 LET SCORE=SCORE+1
  60 POKE 23692,255: REM DESLIZA
R
  70 PRINT INK 7;AT 21,RND*31;"★
";AT 21,RND*31;"★";AT 21,RND*31;
"★"
  80 PRINT AT 10,ATRAVES;" ";AT
21,31''
  90 LET ATRAVES=ATRAVES-(INKEY$
="5" AND ATRAVES>0)+(INKEY$="8"
AND ATRAVES<31)
 100 IF ATTR (10,ATRAVES)=0 THEN
 PRINT AT 10,ATRAVES; INK 4;"V":
 GO TO 50
 110 PRINT AT 0,0; INK 7;"TU PUN
TUACION ";SCORE;AT 10,ATRAVES; I
NK 4; FLASH 1;"V"
 120 INK 9: STOP
1000 REM Grafico "A" estrella
1010 FOR A=0 TO 15
1020 READ UDG
1030 POKE USR "A"+A,UDG
1040 NEXT A
1050 DATA 16,56,254,124,56,108,1
30,0 195,165,90,66,36,36,24,24
1060 RETURN
```

A veces es posible usar el SCREEN$ o el ATTR y es necesario escoger cuál de los dos usar. En situaciones donde se puede usar tanto el uno como el otro, el factor determinante es la velocidad. Pruebe los dos programas siguientes y cronometre cuánto tardan cada uno de ellos en ejecutarse.

```
10 FOR A=1 TO 1000
20 LET B$=SCREEN$ (2,2)
30 NEXT A
```

```
10 FOR A=1 TO 1000
20 LET B=ATTR (2,2)
30 NEXT A
```

El que usa SCREEN$ tarda unos 12 segundos mientras que el del ATTR tarda unos 9 segundos. Así pues, se ve perfectamente que es más rápido volver a los atributos de una posición en la pantalla que retornar el carácter en esa posición. Algunas veces Vd. puede encontrarse en la dificultad de encontrar el color PAPER o el color INK o de si la posición es brillante o destellante. ATTR retorna los atributos de la posición de un carácter como un número del 0 al 255 que consiste en información para cuatro de ellos. Para resolver esto necesitamos saber cómo los atributos están contenidos en la memoria en el fichero de visualización. Este diagrama representa un byte de atributo en la pantalla.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit Ø
flashing	brillo	color PAPER			color INK		

bits 0-2 contienen INK en binario

bits 3-5 contienen PAPER en binario

bit6 contiene el atributo del BRILLO, siendo 1 si brilla (o BRIGHT 1)

bit7 contiene el atributo del DESTELLO siendo 1 si destella (o FLASH 1)

Para resolver estos atributos individuales necesita un poco de ayuda. Las siguientes funciones definidas (FN) resolverán los atributos del FLASH (FN f), el atributo BRIHGT (FN b), el atributo PAPER (FN p) y el atributo INK (FN i). Si Vd. sólo va a usar la expresión una sola vez en un programa no hay ninguna necesidad de fijar las 4 llamadas FN; simplemente escriba la expresión cuando la necesite. Los números que aparecen son de color, así pues, si el color del PAPEL era azul, la función FN p sería el color 1. Los argumentos de las funciones x e y son las coordenadas estándard x e y, de forma que para encon-

trar el color del PAPER de la parte superior izquierda de la pantalla diríamos LET paper = FN p(0,0). x es la columna que va a través de la pantalla y la y es el número de la hilera vertical en la pantalla.

```
  10 DEF FN f(y,x)=INT (ATTR (y,
x)/128)
  20 DEF FN b(y,x)=INT ((ATTR (y
,x)-INT (ATTR (y,x)/128)*128)/64
)
  30 DEF FN p(y,x)=INT ((ATTR (y
,x)-INT (ATTR (y,x)/64)*64)/8)
  40 DEF FN i(y,x)=INT (ATTR (y,
x)-INT (ATTR (y,x)/8)*8)
```

Inevitablemente se hallará en situaciones donde necesite encontrar gráficos definidos por el usuario en la pantalla y el ATTR no puede utilizarse.
Realmente es posible hacer que el SCREEN$ reconozca los gráficos definidos una vez que Vd. entienda como el SCREEN$ funciona. La función SCREEN$ recoge un valor desde la variable de un cierto sistema que le permite hallar el comienzo del conjunto de caracteres el cual busca el patrón de un punto de acoplamiento para el patrón en la pantalla.

Si cambiamos el valor de la variable en este sistema de forma que el SCREEN$ piense que los GDU son los conjuntos de caracteres normales, entonces use una simple operación aritmética para obtener el valor correcto de CHR$ (puede pensar que el GDU A era CHR$ 32 por el contrario, ya que CHR$ 32 es normalmente el primer conjunto de caracteres).

La variable del sistema en cuestión es 23606/23607 que es una variable de dos bytes que contiene un número que es 256 menos que la dirección del comienzo del patrón de puntos del conjunto de caracteres. Hay otra variable del sistema de dos bytes que es la 23675/23676 que nos dice donde comienzan los gráficos definidos por el usuario (exactamente). Así pues, todo lo que necesitamos hacer es transferir los dos bytes desde la 23675/6 a la 23606/7 restando uno desde el byte más alto para la diferencia de 256.
Lo mejor es escribir esto en forma de subrutina. Esta subrutina cambia el puntero del conjunto de caracteres a gráficos definidos por el usuario, revisa la pantalla, toma el valor del CHR$ y después reinicializa el puntero al conjunto de caracteres de la ROM (si Vd. está usando otro conjunto de caracteres tendrá que guardar el valor original en una variable). La X es la posición de PRINT a través de la pantalla y la Y es la posición PRINT a lo largo de la pantalla.

```
8000 POKE 23606,PEEK 23675: POKE
 23607,PEEK 23676-1
8010 LET A$=SCREEN$ (Y,X)
8020 IF A$<>"" THEN LET A$=CHR$
(CODE A$+112)
8030 POKE 23606,0: POKE 23607,60
8040 RETURN
```

Con esta rutina, no se reconoce ningún espacio en blanco en la pantalla a menos que alguno de los gráficos definidos por el usuario se parezca a un espacio o haya algo en la RAM de gráficos definidos por el usuario similar a un espacio. Esto es una molestia teniendo en cuenta que los espacios se usan muchísimo, después de todo cualquier parte en blanco de la pantalla es un espacio. Hay dos formas especiales. O bien definir un GDU como un espacio o añadirlo a la rutina así:

```
8000 LET A$=SCREEN$ (Y,X): IF A$
=" " THEN RETURN
8010 POKE 23606,PEEK 23675: POKE
 23607,PEEK 23676-1
8020 LET A$=SCREEN$ (Y,X)
8030 IF A$<>"" THEN LET A$=CHR$
(CODE A$+112)
8040 POKE 23606,0: POKE 23607,60
8050 RETURN
```

Hacer que el SCREEN$ reconozca los bloques gráficos desde CHR$ 128 a CHR$ 143 no es tarea fácil. No existe ningún patrón de puntos que pueda crear esto ya que están "calculados" cuando han de imprimirse. Vd. podría determinar los patrones de puntos en alguna parte y cambiar el puntero del conjunto de caracteres para apuntarlos o podría definir un gráfico definido por el usuario como el bloque de gráficos y usar esto para revisar el SCREEN$. Recuerde que aunque hay 16 bloques de gráficos, sólo 8 de ellos necesitan ser revisados ya que 8 son versiones en inverso de los otros 8 y el SCREEN$ puede trabajar sobre los inversos como muestra esta sentencia (entrada en modo directo o como una línea de programa):

```
CLS:PRINT CHR$ 143:PRINT SCREEN$
 (0,0),CODE SCREEN$ (0,0)
```

El CHR$ 143 es GRAPHICS SHIFT 8 o un bloque de puntos INK (un espacio inverso). El SCREEN$ reconoce esto como un espacio ordinario CHR$ 32. Naturalmente, una forma fácil de hacerlo es copiar el conjunto de caracteres entero desde la ROM a la RAM y redefinir algunos de los caracteres que el SCREEN$ puede reconocer. Generalmente los caracteres a redefinir suelen ser los menos usados, tales como el cuadrado y los corchetes y otros símbolos no demasiado utilizados en los listados.

LINEAS DE PROGRAMA IMBORRABLES

¿No sería bonito poder insertar una línea como ésta:

```
10 REM © Fred Bloggs 1982
```

en su programa sabiendo que no podrá ser editada ni eliminada por otra gente que no sea el propio autor? Borrar la línea de arriba es fácil: sólo escriba 10 seguido de ENTER y la línea desaparece de la forma más normal. Lo que se necesita es un método de insertar líneas dentro de un listado, las cuales sean muy difíciles, por no decir imposibles, de borrar. La primera parte de la respuesta es que si Vd. tiene un listado en el que hay una línea con el número 0, dicha línea no se puede borrar de ninguna forma normal ya que generalmente la línea número 0 se relaciona con comandos directos (es decir, entrar el comando directo PRINT sin un número de línea: Vd. obtiene 0 OK 0:1 queriendo decir que todo está OK en la primera sentencia de la línea 0). Si Vd. intenta entrar:

```
0 REM © Fred Bloggs 1982
```

le aparecerá un mensaje del BASIC "Nonsense in BASIC" (sin ningún sentido) con el reporte C. Así pues, esto tampoco es válido.

Lo que debe hacer es entrar una línea con un número normal (10, por ejemplo) y entonces cambiar este número por el 0. ¿Difícil?. No demasiado. Esto se hace observando los números de línea del programa, seguidos por un REM, entonces les hace un POKE hasta que consiga lo que quería. Una forma mejor es usar la variable NXTLIN del sistema contenida en la 23637/8, la cual contiene la dirección del comienzo de la siguiente línea del programa (nota: línea, no sentencia). El manual del Spectrum nos informa de que cada línea en BASIC comienza con un número almacenado en dos bytes con el Byte Más Significativo (MSB) seguido por el Byte Menos Significativo (LSB). Así pues, la línea 1 sería 0, 1 y la línea 258 sería 1,2(1*256+2). Entonces, si hacemos un POKE de 0 dentro de ambos bytes conseguiríamos nuestro objetivo de una línea de programa virtualmente no borrable. A continuación verá cómo se hace esto en una línea del BASIC:

```
   1 LET a=PEEK 23637+256*PEEK 2
3638: POKE a,0: POKE a+1,0: STOP
   2 REM © Fred Bloggs
```

Ejecute el programa. Ahora haga un LIST del programa y observe el número de la línea cero donde estaba la línea 2 y observe también como los números de las líneas no han sido clasificados en correcto orden; la clasificación sólo se hace cuando están realmente entrados en el listado – una vez están dentro,están por orden y cualquier línea entrada y las siguientes estarán en su lugar correcto. La línea 1 no se necesita ya más – bórrela para que otros no la puedan usar. Ahora debería tener:

```
0 REM © Fred Bloggs 1982
```

Intente borrarla con el EDIT; escríbalo en su número de línea. Parece bastante fácil,¿no?. Para borrarla tendrá que volver a repetir otra vez el POKE. Pero si se detiene a pensarlo un momento verá que tiene un problema.– Vd. no puede usar la variable del sistema NXTLIN otra vez porque la línea 0 ahora es la primera línea del programa – cualquier otra línea entrada será clasificada e irá a continuación de la línea 0. El NXTLIN sólo dará la dirección correcta para el POKE si lo utilizó antes de la línea que se ha de POKEar – suerte negra. Para seguridad le dejaré adivinar cómo borrar la línea 0. Hay varias formas de hacer esto, todas son "más o menos parecidas" y no demasiado obvias. No hay precio por esto ya que se supone que no debe hacerse. Vd. podría poner todo esto dentro de un programa y si Vd. es lo suficientemente hábil, podría poner la sentencia del copyright brillante, con flash y color en cada página de un listado de manera que pueda apreciarse en cualquier parte. ¡No hay nada más asombroso que un brillante mensaje repartido por toda la pantalla que Vd. está seguro que no podrán borrar!

Si Vd. pretende usarlo muchas veces, podría poner la rutina en la cinta y hacer un MERGE dentro de sus programas. Si empezamos desde el principio, podríamos hacer:

SAVE"copyright" LINE 1

lo que sólo le dejaría con la tarea de borrar la línea 1 ya que la instrucción LINE hará que el programa comience por sí mismo para crear la línea 0. De forma alterna, sólo salve la línea 0 en la cinta entonces use el MERGE para añadir eso como la primera línea del programa. El MERGE contendrá la línea cero. Cuando cree la línea 0 del copyright, le sugiero que le dé el PAPER brillo blanco, el INK negro, y el FLASH.

Incidentalmente, el 0 no es lo único divertido del número de línea que le puede hacer un POKE. Pruebe POKEar algo como el 50 y el 0 en la línea 1. ¿Por qué está el cursor antes del número de línea y en la posición errónea?.

Realmente es un número de línea mayor que 9999, el cual el interpretador no puede decodificar adecuadamente. Para obtener un resultado aún más divertido, intente POKEar el 64 y el 0 en la línea 1 – ¿dónde fue el programa?. Cualquier número de línea mayor que el 16383 hace que el programa no pueda ser listado ya que el LIST no funciona más allá de la línea 16383. El programa todavía está allí pero no puede verse tal como puede comprobar si Vd. vuelve a POKEar para dejar las cosas como estaban.

"PULSE CUALQUIER TECLA PARA CONTINUAR"

Un requerimiento común consiste en suspender la ejecución de un programa a la espera de una respuesta del operador. Un ejemplo sería el de mostrar una lista de instrucciones y decirle al operador que pulse cualquier tecla una vez las haya leído. El siguiente programa puede servir para ello:

```
.....(instrucciones)
1000 PRINT "Pulsa una tecla esce
pto Shift   para continuar"
1010 IF INKEY$="" THEN GO TO 101
0
```

Funciona, pero si Vd. pulsa las teclas CAPS SHIFT o SYMBOL SHIFT, el programa le ignorará olímpicamente y la gente dirá "qué programa más tonto, he pulsado una tecla y no sigue". Hay varias formas de solucionar el problema:

```
.....(instrucciones)

1000 PRINT "Pulsa una tecla para
 continuar"
1010 IF INKEY$="" THEN GO TO 101
0


1000 PRINT "Pulsa ENTER para con
tinuar"
1010 INPUT A$
```

Incidentalmente, Vd. puede haber observado que los programas usan INKEY$, sin embargo, INKEY$ no responde a las teclas SHIFT cuando se pulsan sueltas, sólo reacciona si se pulsan simultáneamente, continuando el programa. La pulsación de ambas teclas a la vez (como cuando se entra en el modo E), produce un CHR$ 14.

Los ejemplos de arriba van, pero ¿no sería mucho mejor que realmente pudiésemos pulsar CUALQUIER TECLA para continuar?. Cualquier tecla significa, por supuesto, una de las 40 que hay en el teclado del Spectrum, incluyendo ambos SHIFTs. He aqui una manera de conseguirlo:

```
1000 PRINT "Pulsa ENTER para con
tinuar"
1010 IF INKEY$="" AND IN 65278=2
55 AND IN 32766=255 THEN GO TO 1
010
```

El teclado está situado en lo que nosotros llamamos espacio I/O (E/S), significando INPUT/OUTPUT (ENTRADA/SALIDA). Estos son métodos para obtener información de la entrada y salida del computador desde y hacia el mundo exterior. Los zócalos MIC y EAR, el altavoz interno, el teclado, la impresora y los microdrives y las interfaces RS232 son todos ejemplos del I/O en acción. La mayor diferencia entre el direccionamiento de la memoria que el usuario conoce como el PEEK y el POKE es que sólo funcionan con la memoria ya sea la RAM o la ROM. Los comandos de I/O IN (ENTRAR) y OUT (SALIR) son los que transmiten información hacia o desde el computador y el mundo exterior. Hay 65536 de estos ports de I/O, lo mismo que 65536 posiciones de memoria hay, pero no todos están en uso, igual que todos los espacios de memoria no se usan en un Spectrum de 16K.

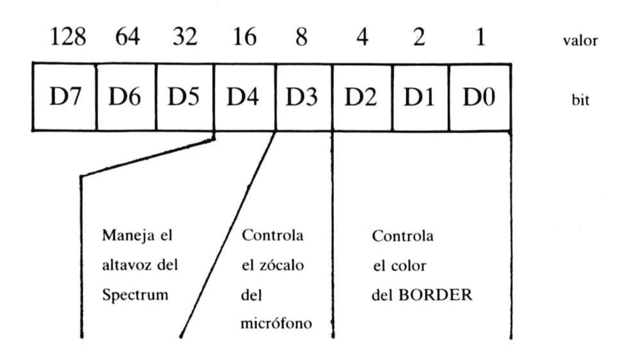
Hay dos comandos en el BASIC que manejan los ports I/O. Son el IN y el OUT que pueden compararse a la forma de trabajar del PEEK y del POKE respectivamente. Las funciones como el INKEY$ también acceden a los ports I/O, pero hacen uso de los códigos de máquina equivalentes al IN y OUT. La siguiente pregunta es cómo Vd. sabe cuál de estos ports son usados para qué. El Capítulo 23 del manual del Spectrum lo define brevemente, pero de los que realmente podemos sacar más partido son los asociados con el teclado, al menos de momento.

Como un ejemplo, usando el OUT, vayamos a jugar con el PORT 254 el cual además de otras muchas cosas determina el color BORDER y el altavoz. Esto puede demostrarse haciendo un RUN de este corto programa:

```
10 OUT 254,INT (RND*256)
20 GO TO 10
```

Vd. debería oír un ruido de clinc en el altavoz del Spectrum y ver como el color BORDER de la pantalla se vuelve loco. El color cambia tan deprisa que verá los varios colores del BORDER a la vez. Observe que mientras este programa se está ejecutando, las dos líneas inferiores de la pantalla no cambian el color (generalmente tienen el mismo color que el BORDER). El BORDER se convierte en el color de la pantalla inferior cuando Vd. escribe algo. Si Vd. entien-

de algo de binario, este diagrama de ocho bits del PORT 254 le ayudará a explicar cómo el port puede hacer más de una cosa a la vez. Igual que los ports I/O de la posición de memoria son bytes de ocho bits.



El D0,D1,D2, etc., significan el bit 0, bit 1, bit 2, etc. Generalmente la D significa DATA, pero ahora esto no debe preocuparnos. Como sólo se usan los bits del 0 al 4, debemos colocar la línea 10 en el anterior programa con:

```
10 OUT 254,INT (RND*32)
20 GO TO 10
```

ya que los bits usados podrían llegar hasta el 0 (el más bajo) y al 31 (el más alto).

A nosotros nos son más útiles los ports de I/O que están asociados con el teclado. Hay ocho ports, cada uno de ellos contiene una hilera de cinco teclas en la mitad izquierda o derecha del teclado. Por ejemplo, el PORT 61438 está asociado con la hilera de cinco teclas, 6 (bit 4), 7 (bit 3), 8 (bit 2), 9 (bit 1) y 0 (bit 0). Pruebe este programa que imprime el valor del port 61438 una y otra vez. Pulse las teclas 6 a la 0 para ver qué efecto tiene. Pulse más de una tecla a la vez.

```
10 PRINT IN 61438
20 PAUSE 100
30 GO TO 10
```

Haga un RUN y vea como imprime 255 todo el tiempo, a menos que Vd. pulse una de las 5 teclas de la mitad de la hilera del 6 al 0 en el teclado. Los números

	PORT BIT					PORT BIT					
	DØ	D1	D2	D3	D4	D4	D3	D2	D1	DØ	
PORT 63486	1 1	2 2	3 4	4 8	5 16	6 16	7 8	8 4	9 2	0 1	PORT 61438
PORT 64510	Q 1	W 2	E 4	R 8	T 16	Y 16	U 8	I 4	O 2	P 1	PORT 57342
PORT 65Ø22	A 1	S 2	D 4	F 8	G 16	H 16	J 8	K 4	L 2	ENTER 1	PORT 4915Ø
PORT 65278	Caps Shift 1	Z 2	X 4	C 8	V 16	B 16	N 8	M 4	Symbol Shift 2	SPACE 1	PORT 32766

valores a sustraer de 225 si se pulsa una tecla

Diagrama de los ports I/O asociados con el teclado y cuyos bits de los ports están asociados con cada tecla. Observe como el bit 0 está siempre afuera y el bit 4 está más cerca del medio del teclado.

obtenidos pueden parecer bastante aleatorios, pero sólo hasta que Vd. comprenda como funciona. Puede haber pensado que 255 es el valor para "ninguna tecla pulsada". También puede saber que 255 en binario es 11111111. Así pues, como los números que obtiene cuando pulsa las teclas son menores que 255, ¿puede imaginarse que pulsando una tecla convierte uno de estos unos binarios en un cero?.

Estudie este diagrama (p. 88) que muestra los bits de los ports relacionados con las teclas correspondientes. En particular, intente estudiar estas teclas que ha estado usando como ejemplos, de la 6 al 0.

Haga un RUN del programa otra vez y cada vez que pulse una tecla, reste a 255 el número escrito debajo de las teclas del teclado en el diagrama, o sea, si Vd. está pulsando el 0 reste 1 de 255, dándole 254. Si Vd. está pulsando la tecla 8, reste 4 de 255, dándole 251, y así sucesivamente. Debería obtener el mismo número a medida que el programa escribe en la pantalla.

Puede que esto no tenga mucho sentido al principio, pero persevere porque todo se aclarará a lo largo de este manual. Escrito sobre las teclas en el diagrama aparecen los símbolos D0 al D4 otra vez – éstos representan bits individuales del port I/O. En esta aplicación sólo los bits del 0 al 4 se usan para el teclado, ya que sólo son cinco los que cada port tiene que revisar.

Veamos algunos ejemplos para demostrar un simple uso del IN para buscar el teclado.

Para verificar si se ha pulsado la tecla R:

```
IF IN 64510=(255-8) THEN PRINT "R
 ES PRESIONADA"
```

Para revisar si se ha pulsado la tecla Y:

```
IF IN 57342=(255-16) THEN PRINT
"Y ES PRESIONADA"
```

Para revisar si se ha pulsado la tecla ESPACIO:

```
IF IN 53742=(255-1) THEN PRINT "S
PACE ES PRESIONADA"
```

Naturalmente, Vd. no tiene que escribir la expresión encerrada entre paréntesis como en los ejemplos de arriba –han sido escritos para ilustrar el punto que

Vd. resta a 255 el valor del bit. Observe que si Vd. añade todos los valores de los bits juntos, la respuesta es la misma. En este momento no causa ninguna diferencia de como lo hace. Ahora es más importante un resultado correcto y entenderlo. Conviene observar que cualquier bit es sólo un cero si se pulsa la tecla correspondiente. Esto explica por qué obtenemos un valor de 255 si no se pulsa nada – todos los bits son 1, de manera que el total en decimal es 255. Tomemos como ejemplo cuando se pulsa la tecla K. El port I/O asociado con la mitad de la hilera de 5 teclas es 49150 (ver diagrama del teclado). Cada byte o port tiene ocho bits, así:

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
1	1	1	1	1	1	1	1

Lo de arriba nos muestra la mitad de la hilera sin haber pulsado ninguna tecla.

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
1	1	1	1	1	0	1	1

Ahora tiene un valor de BIN 11111011 que es (en decimal) (255–4) o 251, que es también lo mismo que (128+64+32+16+8+2+1). Técnicamente, la forma de hacerlo correctamente es añadir los bits individualmente, pero el otro método también funciona por razones que no vamos a entrar aquí, y que generalmente es más fácil de usar para esta aplicación. Vd. podría hacer lo mismo para cualquier otra tecla del teclado. Así:

```
IF IN 49150=251 THEN PRINT"K ES
PRESIONADA"
```

No gana nada respecto a:

```
 IF INKE$=K OR INKEY$=k THEN PRI
NT"K ES PRESIONADA"
```

Sin embargo, tiene sus ventajas. Por ejemplo, Vd. puede comprobar si algunas de las SHIFTs están pulsadas, cosa que no podría hacer con el INKEY$, o sea:

```
IF IN 65278=254 OR IN 32766=253
"THEN PRINT MAYUSCULAS"
```

El INKEY$ también distingue entre letras en mayúsculas y minúsculas de manera que IF INKEY$ = "k" THEN... no es lo mismo que IF INKEY$ = "K" THEN... donde IF IN 49150 = 251 THEN... sólo comprueba que se ha pulsado la k, tanto si el CAPS LOCK está activado como si no.

El uso del IN para buscar en el teclado también nos permite revisar si se ha pulsado más de una tecla o combinaciones de teclas, es decir:

```
IF IN 49150=(255-2-4) THEN PRINT
"K Y L PRESIONADAS"
```

Una aplicación de esto serían los juegos donde las teclas de control del cursor se usan para controlar el movimiento en la pantalla en la dirección de las flechas. La mayoría de los juegos sólo le permiten moverse a la izquierda, abajo, arriba a la derecha, pero nunca en diagonal. Usando el IN podríamos ver si ambas teclas la 5 y la 6 están pulsadas para poder hacer el movimiento de a la izquierda y abajo, o sea, en diagonal en dirección a la parte inferior izquierda de la pantalla de manera que el control del movimiento se pareciese al de un joystick. Pruebe este programa para dibujar una línea en dirección hacia arriba y a la derecha desde la parte inferior izquierda de la pantalla en dirección a la parte superior derecha, algo así como un rasgo. Los controles son el 7 para mover hacia arriba, el 8 para mover a la derecha y pulsar el 7 y el 8 para mover en diagonal y a la derecha. Esto no nos hubiera parecido tan fácil si hubiéramos usado el INKEY$ ya que no habría manera de saber si ambas teclas la 7 y la 8 están pulsadas a la vez.

```
10 LET X=0
20 LET Y=0
30 PLOT X,Y
40 LET A=IN 61438
50 LET X=X+(A=251 OR A=243)
60 LET Y=Y+(A=247 OR A=243)
70 GO TO 30
```

Mientras que con este método usamos las teclas de cursor 5, 6, 7 y 8 para controlar los movimientos de la pantalla, ¿no sería mejor usar otro método que fuera más sencillo?. Las razones que se usan comúnmente para esta finalidad son que las teclas de dirección están juntas en el teclado y son fáciles de leer con INKEY$ para controlar los valores de las variables (Vd. puede estar familiarizado con el LET X=X+ (INKEY$="8")-(INKEY$="5")). El problema es que estas teclas están tan cerca unas de otras que se necesita una combinación

de dedos muy complicada, para tener el control. El sistema a describir nos permite el uso de las 40 teclas del teclado para controlar el movimiento, de forma que Vd. no tiene que preocuparse demasiado de si sus dedos pulsan las teclas adecuadas. El teclado se dividirá en 4 partes (desde el punto de vista del programa), cada una de ellas de un bloque de 10 teclas como ésta controlando el movimiento en las direcciones que se muestran.

1			**ARRIBA**		0
Q	**IZQUIERDA**	T	Y	**DERECHA**	P
A	**IZQUIERDA**	G	H	**DERECHA**	ENTER
CAPS SHIFT			**ABAJO**		ESPACIO

Así pues, pulsando cualquier tecla de la fila de teclas de la parte superior del teclado produce el movimiento hacia arriba, pulsando cualquiera de las teclas de la hilera de teclas de la parte inferior del teclado, el movimiento es hacia abajo. Pulsando las teclas de la mitad izquierda de la hilera del teclado hace que se mueva a la izquierda y pulsando la otra mitad de la derecha del teclado, su movimiento es a la derecha. Pulsando diferentes grupos de teclas tienen un efecto combinado, o sea, si Vd. pulsó la tecla 3 y la tecla W se moverá en diagonal desde arriba a la izquierda. El programa que demuestra la rutina consiste en un sencillo dibujador que traza una línea en la dirección que Vd. le marca. Si no pulsa ninguna tecla, el trazo permanece quieto, como es de esperar. No se piense que éste es el mejor programa de dibujo de todos los tiempos

(¡se rompe si Vd. se sale de la pantalla!). Vea el diagrama anterior que muestra los ports I/O asociados con el teclado cuando examine las teclas 30 y 40 que son las que hacen todo el trabajo de escudriñar el teclado.

```
   10 LET X=120
   20 LET Y=90
   30 LET X=X+(IN 49150<>255 OR I
N 57342<>255)-(IN 64510<>255 OR
IN 65022<>255)
   40 LET Y=Y-(IN 65278<>255 OR I
N 32766<>255)+(IN 63486<>255 OR
IN 61438<>255)
   50 PLOT X,Y
   60 GO TO 30
```

IMPRESION DE MATRICES DE CADENAS ALFANUMERICAS

Pruebe este programa:

```
   10 DIM A$(12,9)
   20 FOR A=1 TO 12
   30 READ A$(A)
   40 PRINT A$(A);",";
   50 NEXT A
   60 PRINT CHR$ 8;"."
   70 DATA "ENERO","FEBRERO","MAR
ZO","ABRIL","MAYO","JUNIO","JULI
O","AGOSTO","SEPTIEMBR ","OCTUBR
E","NOVIEMBRE","DICIEMBRE"
```

Lo que debe hacer es asignar los nombres de los meses del año para cada cadena de la matriz (12 meses, longitud máxima de 9 letras – DIM A$(12,9)) e imprimirlas de la primera a la última en una hilera, separadas por comas, terminando con un punto. Así es como deberán aparecer:

```
ENERO    ,FEBRERO  ,MARZO    ,AB
RIL      ,MAYO     ,JUNIO    ,JULI
O    ,AGOSTO   ,SEPTIEMBR ,OCTUBR
E   ,NOVIEMBRE,DICIEMBRE.
```

Algunas de las palabras están separadas por varios espacios cuando se imprimen porque cada cadena en la matriz A$ siempre tiene la misma longitud (en este caso nueve caracteres). Cada vez que Vd. la asigna a uno de los elementos, Vd. asigna todos los caracteres de la cadena,aun cuando el computador tiene que añadir espacios extra. Por ejemplo, si Vd. está haciendo A$(5)MAYO que consiste en sólo cuatro letras, después deben añadirse cinco espacios para completar los nueve caracteres. El resultado es la impresión de una serie de espacios, muy desagradables, que aparecen además de las letras

(1) Imprima la cadena, buscando hacia atrás para encontrar el final de la palabra, así:

```
   10 DIM A$(12,9)
   20 FOR A=1 TO 12
   30 READ A$(A)
   40 REM IMPRI. Y OMIT. ESPACIOS
   50 FOR B=LEN A$(A) TO 1 STEP -
1
   60 IF A$(A,B)<>" " THEN PRINT
A$(A, TO B);",";: GO TO 80
   70 NEXT B
```

```
 80 NEXT A
 90 PRINT CHR$ 8;"."
100 DATA "ENERO","FEBRERO","MAR
ZO","ABRIL","MAYO","JUNIO","JULI
O","AGOSTO","SEPTIEMBR ","OCTUBR
E","NOVIEMBRE","DICIEMBRE"
```

Este será el resultado de la ejecución del programa:

```
ENERO,FEBRERO,MARZO,ABRIL,MAYO,J
UNIO,JULIO,AGOSTO,SEPTIEMBR,OCTU
BRE,NOVIEMBRE,DICIEMBRE.
```

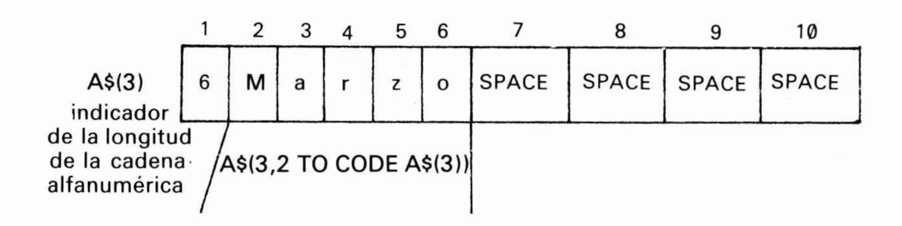
Todo lo que el programa hace es colocar los nombres de los doce meses dentro de la matriz A$(12,9) y la impresión se hace en el bucle FOR–NEXT B, el cual funciona hacia atrás desde el último carácter de cada cadena hasta que encuentra un carácter que no tenga ningún espacio. Entonces lo imprime con el A$(A, TO B). Observe la coma. Esta expresión es una abreviatura del A$(1 TO B).

(2) Use el indicador de Longitud de la Cadena (SLI) para registrar la longitud de las palabras en las cadenas de forma que no necesite perder el tiempo buscando antes de imprimir. Este método también sirve para palabras que finalicen con espacios, se impriman correctamente (si asi se desea) ello se muestra en el siguiente listado:

```
 10 DIM A$(12,10)
 20 FOR A=1 TO 12
 30 READ S$
 40 LET A$(A)=CHR$ (LEN S$+1)+S
$
 50 PRINT A$(A,2 TO CODE A$(A))
;",";
 60 NEXT A
 70 PRINT CHR$ 8;"."
 80 DATA "ENERO","FEBRERO","MAR
ZO","ABRIL","MAYO","JUNIO","JULI
O","AGOSTO","SEPTIEMBR ","OCTUBR
E","NOVIEMBRE","DICIEMBRE"
```

Observe que la línea 10 tiene un elemento extra, el 10, que se ha añadido en cada cadena de la matriz A$, para almacenar el SLI. El número que representa la longitud de la cadena debe de ser un número entre 0 y 255. Eso debe de bastar para la mayoría de aplicaciones. Podrían utilizarse dos bytes si los números más largos llegaran hasta el 65535. El bucle A lee los nombres de los meses desde la lista de datos dentro de una variable de cadena común S$. Como las variables se alargan según lo necesario, podemos usar el LEN S$ para encontrar la longitud de la palabra y almacenarla en el primer carácter de las cadenas

de la matriz A$ como CHR$(LEN S$+1). Este es el elemento extra del que hablamos anteriormente. El resto de la cadena se hace dentro de una copia del S$. Así es como aparecería para el mes de Marzo:



Así pues, el primer elemento de la cadena A$(3) contiene un carácter 6. La longitud de la palabra Marzo tiene 5 letras. Comienza en el segundo elemento y termina en el elemento 6. La razón de añadir un 1 a la longitud de Marzo es debida a que el SLI es el primer carácter,– por el contrario se colocaría al final en el caso de que la cadena asignada fuera demasiadoo larga y sobrepasara el SLI. El único problema es que el SLI puede ser mayor que la longitud de las cadenas del A$ generando un error de subíndice. Esto podría evitarse añadiendo la línea:

```
45 IF CODE A$(A)>=LEN A$(A) TH
EN LET A$(A,1)=CHR$ (LEN A$(A)-1
)
```

Las cadenas se imprimen como PRINT A$(A,2 TO CODE A$(A)), significando desde el carácter que sigue al SLI hasta el último que no tiene un espacio. Una gran ventaja del método (2) sobre el método (1) es que Vd. puede perfectamente imprimir finales de palabras con espacios y si quiere copiar cadenas dentro de otras cadenas o simplemente almacenar el SLI para hacerlo más rápido.

Si Vd. quisiera la velocidad y conveniencia de los SLI pero sin la pérdida de las cadenas, entonces simplemente debe almacenar los SLI en una matriz de cadenas o numéricas. Primero la matriz de cadenas:

```
 1 REM USO DE MATRIZ EN FILA
10 DIM A$(12,9)
20 DIM B$(12)
30 FOR A=1 TO 12
4^ READ S$
```

```
  50 LET A$(A)=S$
  60 LET B$(A)=CHR$ (LEN S$)
  70 PRINT A$(A, TO CODE B$(A));
",";
  80 NEXT A
  90 PRINT CHR$ 8;"."
 100 DATA "ENERO","FEBRERO","MAR
ZO","ABRIL","MAYO","JUNIO","JULI
O","AGOSTO","SEPTIEMBR ","OCTUBR
E","NOVIEMBRE","DICIEMBRE"
```

Y con una matriz numérica:

```
   1 REM USO DE MATRIZ NUMERICA
  10 DIM A$(12,9)
  20 DIM B(12)
  30 FOR A=1 TO 12
  40 READ S$
  50 LET A$(A)=S$
  60 LET B(A)=LEN S$
  70 PRINT A$(A, TO B(A));",";
  80 NEXT A
  90 PRINT CHR$ 8;"."
 100 DATA "ENERO","FEBRERO","MAR
ZO","ABRIL","MAYO","JUNIO","JULI
O","AGOSTO","SEPTIEMBR ","OCTUBR
E","NOVIEMBRE","DICIEMBRE"
```

Como habrá comprobado, el uso de una matriz numérica es más rápido pero gasta más memoria. Nuevamente, esto se convertiría en una rutina para usar dentro de los programas.

ATRIBUTOS DE LA PANTALLA INFERIOR

Generalmente Vd. no puede cambiar los atributos de la pantalla inferior, excepto para el color del PAPER, el cual sigue al color del BORDER. ¿Le agradaría obtener un reporte con el código y cursor en verde?. Se puede hacer. La variable del sistema 23624 contiene los atributos utilizados para la pantalla inferior y el color del BORDER.

Aquí viene un diagrama para mostrar la función de cada bit de esta variable del sistema, llamada BORDER:

bit	7	6	5	4	3	2	1	Ø
	FLASH de la parte inferior de la pantalla	BRIGHT de la parte inferior de la pantalla	Color BORDER y PAPER parte inferior de la pantalla			Color INK de la parte inferior de la pantalla		

Mediante un POKE de diversos valores en esta variable del sistema se puede conseguir, por ejemplo, un negro destelleante y una parte baja de la pantalla en verde o bien que dicha parte sea blanca pero más brillante que el resto de la pantalla, en blanco tambien para las sentencias INPUT, etc. Pruebe los dos que siguen con una pantalla blanca (PAPER 7:CLS):

```
POKE23624,BIN 10111000     (184)
POKE23624,BIN 01111000     (120)
```

Este efecto no se puede conseguir normalmente con las sentencias INPUT que afecten sólo al mensaje de dichas sentencias. El cambio del color BORDER lo afecta; por ejemplo, la instrucción INK de la parte baja de la pantalla (que es "automáticamente" 9), revertería en blanco o negro para asegurar el máximo contraste,de modo que cualquier cosa que se escriba en la pantalla se pueda leer fácilmente.

COMO PREVENIR LA EJECUCION AUTOMATICA

Si un programa se ha grabado en la cinta usando la facilidad SAVE "prog" LINE X, ese programa comenzará automáticamente cuando lo cargue otra vez en el computador, comenzando desde la línea X. Para evitar que esto ocurra si desea hacer un MERGE"", asegúrese de que anteriormente no había ningún programa en el computador antes de usar el MERGE"", ya que puede obtener una combinación de ambos programas.

COMO HACER MAS RAPIDOS SUS PROGRAMAS

Hay algunos trucos para ayudarle a conseguir que los programas en BASIC del Spectrum corran el máximo posible. El uso de estas técnicas le capacitarán para convertir un lento programa gráfico en una versión más aceptable. Desde luego que los peores programas son aquellos que están pobremente estructurados. Un lío de GOTOs entremezclados, ¡a menudo ralentizarán su programa tanto como si hubiesen unas cuantas sentencias PAUSE!. Lo primero que debe de hacer es organizar sus programas, de modo que se puedan identificar bloques completos como rutinas separadas. Ello no sólo hace que el programa sea de más fácil lectura sino que también es posible el modificar una rutina en particular sin alterar el funcionamiento del resto. Así, los programas no perderán tiempo saltando de una parte a otra salvo que sea estrictamente necesario. De a su programa una buena estructura. Examine todos los GOTOs no condicionales e intente construir el programa mediante una serie de subrutinas que se invoquen desde un bloque principal, como por ejemplo:

```
10 REM PROGRAMA PRINCIPAL


......................


4000 STOP
5000 REM INICIALIZACION
5900 RETURN
6000 REM CONJUNTO DE GRAFICOS
6900 RETURN
          etc. etc.
```

Lo que tiene que observar aquí es que cuando el computador busca destinos GOTO o GOSUB, comienza por el principio del programa y comprueba los números de línea hasta que se encuentra el de destino. Esto significa que si Vd. llamó una subrutina al final de un programa largo varias veces durante el curso del mismo, tardaría más que si esa subrutina estuviera al comienzo del programa. Podemos usar estos programas simples para ilustrarlo:

PROGRAMA 1 – 9 segundos

```
10 FOR A=1 TO 1000
20 GO SUB 9000
30 NEXT A
```

```
  40 STOP
  50 REM
  60 REM
  70 REM
  80 REM
  90 REM
 100 REM
 110 REM
 120 REM
 130 REM
 140 REM
 150 REM
 160 REM
 170 REM
 180 REM
 190 REM
 200 REM
9000 RETURN
```

PROGRAMA 2 – 7.5 segundos

```
  10 FOR A=1 TO 1000
  20 GO SUB 9000
  30 NEXT A
  40 STOP
9000 RETURN
```

La ejecución del programa 1 tarda unos 9 segundos comparado con los 7.5 segundos del programa 2. La razón por la que el programa 1 tarde más es que para encontrar la subrutina en la línea 9000 tiene que saltar sobre todas las líneas una por una. Observe que el número de línea no tiene ningún efecto sobre la velocidad, sólo su posición con el programa. De aquí podemos deducir que los programas deben de diseñarse de modo que los GOTOs y GOSUBs más comúnmente utilizados se posicionen de la siguiente manera:

```
   1 REM Diseño sugerido del programa
  10 GOTO 1000: REM saltar sobre las subrutinas
 100 REM Subrutinas más frecuentes
1000 REM programa principal
8000 REM Subrutinas menos frecuentes.
```

Este diseño sólo se usará cuando el tiempo ahorrado sea importante. Como la mayoría de estas técnicas que ahorran tiempo sólo se convierten en importantes durante los bucles largos. Aquí es donde Vd. va a necesitar más ahorrar tiempo. Ya que estamos hablando de bucles, hay dos tipos principales de bucles que realizan una función similar; los bucles FOR/NEXT y los bucles IF... THEN GOTO... Comparemos la velocidad de ambos.

PROGRAMA 3 – 9 segundos

```
10 LET A=1
20 LET A=A+1
30 IF A<=1000 THEN GO TO 20
```

PROGRAMA 4 – 4.5 segundos

```
10 FOR a=1 TO 1000
20 NEXT a
```

Así pues, el bucle FOR/NEXT es aproximadamente el doble de rápido que el bucle IF...THEN GOTO. Como una comparación interesante, compare el programa 4 con el 5, el cual utiliza sentencias múltiples.

PROGRAMA 5 – 4.5 segundos

```
10 FOR a=1 TO 1000: NEXT a
```

Observe que al contrario que la mayoría de las versiones del BASIC, el del Spectrum no gana mucho tiempo con el uso de sentencias múltiples comparado con el mismo programa escrito con una línea individual para cada sentencia. El programa 5 se toma aproximadamente el mismo tiempo que el 4. No obstante, con las sentencias PRINT, la unión de varias sentencias en una línea puede correr más que con líneas separadas.

PROGRAMA 6 – 45 segundos

```
10 FOR A=1 TO 1000
20 PRINT AT 0,0;"PARECE"
30 PRINT AT 0,0;"PARECE"
40 PRINT AT 0,0;"PARECE"
50 PRINT AT 0,0;"PARECE"
60 NEXT A
```

PROGRAMA 7 – 41 segundos

```
  10 FOR A=1 TO 1000
  20 PRINT AT 0,0;"PARECE";AT 0,
0;"PARECE";AT 0,0;"PARECE";AT 0,
0;"PARECE"
  30 NEXT A
```

Cuando Vd. quiere mover la posición PRINT a una nueva línea, el uso del apóstrofe en lugar de TAB 0; es más rápido.

PROGRAMA 8 – 23 segundos

```
10 FOR A=1 TO 300
20 POKE 23692,255
30 PRINT "+";TAB 0;
40 NEXT A
```

PROGRAMA 9 – 18 segundos

```
10 FOR A=1 TO 300
20 POKE 23692,255
30 PRINT "+"'
40 NEXT A
```

La utilización del carácter de control CHR$13 (el carácter ENTER) para producir un retorno de carro en lugar del apóstrofe no produce ningún cambio en los tiempos de ejecución. La inclusión de los caracteres de control hacen que las cosas vayan más despacio. El programa 10 llena la pantalla con signos + . El programa 11 llena la pantalla con signos + inversos,el cual debe entrarse como "INV.VIDEO + TRUE VIDEO".

PROGRAMA 10 – 4.7 segundos

```
10 FOR A=1 TO 704
20 PRINT "+";
30 NEXT A
```

PROGRAMA 11 – 5.5 segundos

```
10 FOR A=1 TO 704
20 PRINT "+";
30 NEXT A
```

Sin embargo, compare el programa 11 con el 12, el cual usa la característica del INVERSE (INVERSO). Este hace que vaya más despacio que cuando se incluye el carácter de control del video inverso en la cadena. Experiméntelo con otros como el FLASH o BRIGHT.

PROGRAMA 12 – 6.2 segundos

```
10 FOR A=1 TO 704
20 PRINT INVERSE 1;"+";
30 NEXT A
```

Algunas computadoras tienen variables especiales llamadas variables de enteros,las cuales sólo pueden almacenar números sin decimales. Pueden manejarse mucho más deprisa que las variables de coma flotante normales. En el Spec-

trum no existen estas variables de enteros, pero tiene una representación especial para enteros pequeños, el cual se utiliza automáticamente para números que van desde –65535 al +65535 y no tienen decimales.

PROGRAMA 13 – 11.3 segundos

```
 5 POKE 23692,255
10 FOR A=1 TO 600
20 PRINT A;
30 NEXT A
```

PROGRAMA 14 – 15.3 segundos

```
 5 POKE 23692,255
10 FOR A=1.3 TO 600.3
20 PRINT A;
30 NEXT A
```

Los programas del 13 al 18 intentan demostrar como usando los enteros pueden hacerse programas más rápidos. Vd. no tiene ningún control sobre esta acción automática del "entero", así pues, lo mejor que puede hacer es asegurarse de no generar ningún número que no sea entero que hagan las cosas más lentas sin que Vd. lo pueda percibir.

PROGRAMA 15 – 12.1 segundos

```
 5 POKE 23692,255
10 FOR A=1 TO 600
20 PRINT 1.3;
30 NEXT A
```

PROGRAMA 16 – 9.1 segundos

```
 5 POKE 23692,255
10 FOR A=1 TO 600
20 PRINT 1;
30 NEXT A
```

PROGRAMA 17 – 12.3 segundos

```
 5 POKE 23692,255
 7 LET B=1.3
10 FOR A=1 TO 600
20 PRINT B;
30 NEXT A
```

PROGRAMA 18 – 9.2 segundos

```
 5 POKE 23692,255
 7 LET B=1
10 FOR A=1 TO 600
20 PRINT B;
30 NEXT A
```

Vd. debería usar enteros donde sea posible ya que pueden imprimirse o ser convertidos a otro formato más rápidamente que con números del punto flotante usuales. Recuerde que en este formato pueden manejarse las constantes y las variables. También, la cantidad de deslizamientos de la pantalla que tienen que hacerse en los programas de arriba tienen un pequeño efecto en los tiempos de ejecución,ya que la parte inferior de la pantalla se alcanza más rápidamente con números con más dígitos, y los números del punto flotante tendrán más dígitos que las partes del contador de su entero.

Permanezcamos con impresión en pantalla y veamos la forma de imprimir números en la misma. Los programas 19, 20 y 21 comparan las constantes de la cadena de impresión, las constantes numéricas de enteros y constantes numéricas no enteras.

PROGRAMA 19 – 9.8 segundos

```
 5 POKE 23692,255
10 FOR A=1 TO 1000
20 PRINT "69";
30 NEXT A
```

PROGRAMA 20 – 17.2 segundos

```
 5 POKE 23692,255
10 FOR A=1 TO 1000
20 PRINT 69;
30 NEXT A
```

PROGRAMA 21 – 24.8 segundos

```
 5 POKE 23692,255
10 FOR A=1 TO 1000
20 PRINT 69.25;
30 NEXT A
```

Ahora hagamos lo mismo con las variables. Los programas 22, 23 y 24 imprimen las variables de cadena, las variables numéricas de enteros y las variables numéricas no enteras respectivamente.

PROGRAMA 22 – 10.3 segundos

```
 5 POKE 23692,255
 7 LET B$="69"
10 FOR A=1 TO 1000
20 PRINT B$;
30 NEXT A
```

PROGRAMA 23 – 17.9 segundos

```
 5 POKE 23692,255
 7 LET B=69.25
10 FOR A=1 TO 1000
20 PRINT B;
30 NEXT A
```

PROGRAMA 24 – 25.2 segundos

```
 5 POKE 23692,255
 7 LET B=69
10 FOR A=1 TO 1000
20 PRINT B;
30 NEXT A
```

Otra vez, el mensaje es: use las cadenas para imprimir cuando sea posible. Si tiene que ser un número, asegúrese de que es un entero y se imprimirá más rápidamente. Veamos lo que nos indica el VAL y el CODE en la impresión.

PROGRAMA 25 – 26.1 segundos

```
 5 POKE 23692,255
10 FOR A=1 TO 1000
20 PRINT VAL "69";
30 NEXT A
```

PROGRAMA 26 – 18.3 segundos

```
 5 POKE 23692,255
10 FOR A=1 TO 1000
20 PRINT CODE "E";
30 NEXT A
```

Así pues, cuando desee almacenar información en forma de caracteres en una cadena alfanumérica, será mejor decodificarla usando el CODE (CODIGO) en lugar del VAL.

El uso de nombres de variables con múltiples caracteres es más lento que los nombres con un solo carácter.

PROGRAMA 27 – 6.9 segundos

```
10 FOR A=1 TO 1000
20 LET M=10
30 NEXT A
```

PROGRAMA 28 – 7.8 segundos

```
10 FOR A=1 TO 1000
20 LET LIBRO=10
30 NEXT A
```

Primero defina las variables utilizadas más a menudo; el tiempo en encontrar una variable en el área de las variables depende de lo lejos que el computador tenga que llegar mirando esta área.

PROGRAMA 29 – 8.4 segundos

```
10 LET B=10
20 LET C=20
30 LET D=4
40 LET E=6
50 LET F=7
60 FOR A=1 TO 1000
70 LET G=B
80 NEXT A
```

PROGRAMA 30 – 8.8 segundos

```
10 LET B=10
20 LET C=20
30 LET D=4
40 LET E=6
50 LET F=7
60 FOR A=1 TO 1000
70 LET G=F
80 NEXT A
```

Veamos las sentencias REM. Aunque éstas son ignoradas durante la ejecución de un programa, el interpretador del BASIC todavía tiene que pasar por ellas y esto tarda una infinidad de tiempo. Aquí o allí una sentencia REM no variará demasiado, pero puede tener un efecto notable con un bucle largo. Compare el programa 4 con el programa 31:

PROGRAMA 31 – 5.2 segundos

```
10 FOR A=1 TO 1000
20 REM ES UN REM DE INFORMACIO
N
30 NEXT A
```

El programa 4, un bucle FOR/NEXT vacío tarda 4.5 segundos para hacer un RUN comparado con los 5.2 segundos que tarda con una sentencia REM. Esto siempre sucede con los bucles – si puede ponerse fuera del bucle, el computador no perderá el tiempo repitiéndolo una y otra vez.

Veamos varias formas de escribir expresiones que se usan varias veces durante un programa. Veremos el ejemplo de LET R=INT(RND*9)+1. Primero escribiéndolo entero cada vez que se necesite.

PROGRAMA 32 – 13.3 segundos

```
10 FOR A=1 TO 500
20 LET R=INT (RND*9)+1
30 NEXT A
```

O definiendo una función.

PROGRAMA 33 – 13.9 segundos

```
 5 DEF FN R()=INT (RND*9)+1
10 FOR A=1 TO 500
20 LET R=FN R()
30 NEXT A
```

O poniendo la expresión en una subrutina.

PROGRAMA 34 – 14.9 segundos

```
10 FOR A=1 TO 500
20 GO SUB 50
30 NEXT A
40 STOP
50 LET R=INT (RND*9)+1
60 RETURN
```

O poniendo el VAL en una cadena que contiene la expresión.

PROGRAMA 35 – 19.6 segundos

```
 5 LET A$="INT (RND*9)+1"
10 FOR A=1 TO 500
20 LET A=VAL A$
30 NEXT A
```

Ahora Vd. puede ver que es más rápido escribir la expresión completa afuera cada vez que se necesita.

Ahora veamos el ATTR y SCREEN$. Si Vd. puede escoger entre SCREEN$ y ATTR para encontrar lo que hay en cualquier posición de la pantalla, es más rápido usar el ATTR que el SCREEN$, como lo muestran los programas 36 y 37.

PROGRAMA 36 – 9.5 segundos

```
10 FOR A=1 TO 1000
20 LET B=ATTR (1,1)
30 NEXT A
```

PROGRAMA 37 – 12.5 segundos

```
10 FOR A=1 TO 1000
20 LET B$=SCREEN$ (1,1)
30 NEXT A
```

COMO USAR LAS VARIABLES DEL SISTEMA

Las variables del sistema son bytes en la memoria desde el 23552 al 23732 que ayudan al computador a recordar ciertas cosas que necesita saber sobre sí mismo de como está organizada su memoria. La información está contenida en estas variables del sistema en estas direcciones de manera que el computador puede mantenerlas y actualizarlas cuando lo necesite.

Nosotros podemos usar esta información almacenada en estas posiciones de la memoria, de varias formas, en algunos programas tanto para leer la información que ya está almacenada allí o cambiándola para que el computador haga algo que de otra forma no podría hacer, o algunas veces hacerlo más fácil.

No todas estas formas nos son útiles. Y verdaderamente no podemos cambiar todas ellas. Algunas harán que el computador "se cuelgue", o que simplemente las ignore. Algunas pueden cambiarse sin ningún problema pero siempre bajo ciertas circunstancias solamente y la mayoría con limitaciones muy estrictas. Espero poderle facilitar una buena guía de lo que puede hacerse y de lo que no, pero Vd. también aprenderá a su tiempo sus propios pequeños PEEKs y POKEs.

23552 al 23559 KSTATE lectura del teclado

Cuando se interrumpe el procesador (generalmente unas 50 veces cada segundo en Europa), una de las cosas a hacer es leer el teclado y almacenar los resultados allí. Los bytes tienen usos diferentes. Observe que casi todos pueden ser utilizados por el programador. Vd. puede usar este programa para examinar lo que sucede en los ocho bytes del KSTATE. Ejecútelo y pulse varias teclas para ver qué efecto tienen las teclas individuales, como las del SHIFT, y qué efecto tiene yendo desde una tecla a otra.

```
  10 FOR A=23552 TO 23559
  20 POKE 23692,0: REM MANTENER
SCROLL
  30 LET B=PEEK A
  40 PRINT A;TAB 10;B;TAB 20;CHR
$ B AND B>31
  50 NEXT A
  60 GO TO 10
```

Los primeros cuatro bytes del KSTATE tratan con algo llamado "Sobrepulsación de dos teclas" que le permite pulsar una segunda tecla antes de que Vd.

salga de la primera. Las descripciones dadas a los cuatro bytes principales, 23556 al 23559, también se aplicarán a los primeros cuatro mientras Vd. mantenga en mente que sólo se recuerda la pulsación de dos teclas. El PEEK 23556 puede retornar el CODE (CODIGO) de la versión de letras en mayúscula de la tecla pulsada, de forma que si Vd. pulsó SYMBOL SHIFT A, obtiene el CODIGO de "A" no el CODIGO de "a" ni de "STOP". Esto puede serle útil donde sea necesario entrar en letras mayúsculas, etc. El efecto de pulsar una tecla es temporal y sólo dura mientras está pulsando. El valor en el 23556 sería 255 si no se pulsara ninguna tecla en el momento de la interrupción. Para la tecla ENTER se retorna un valor de 13. Para la tecla SPACE (ESPACIO) es retornado un valor de 32. Pulsando ambas teclas simultáneamente produce el 14. Este programa se lo demostrará:

```
10 LET A=PEEK 23556
20 POKE 23692,0
30 PRINT A,CHR$ A AND A>31
40 GO TO 10
```

El 23557 se usó para prevenir el contacto de una tecla intermitente, etc., causando problemas, conocido como "efecto de rebote".

El 23559 contiene el CODE (CODIGO) del último carácter pulsado en el teclado. Esto depende de si se pulsaron o no las teclas SHIFT. Los números que aparecen son los mismos que hubiera producido el PRINT CODE INKEY$ excepto que éstos son las últimas teclas pulsadas, no necesariamente la actual tecla pulsada. Pruebe este programa para visualizar lo que sucede. Ejecútelo e intente pulsar varias teclas haciendo el uso de las teclas SHIFT.

El 23558 controla el tiempo que pasa desde que se pulsa una tecla hasta que empieza la autorepetición. Cada 50 unidades es un segundo.

```
10 LET A=PEEK 23559
20 POKE 23692,0
30 PRINT A,CHR$ A AND A>31
40 GO TO 10
```

Vea también los FLAGS del 23611.

23560 LASTK Tecla pulsada más Recientemente

Cada vez que se rastrea el teclado y se encuentra una tecla pulsada cuyo valor se ha comprobado como válido, se actualiza esta variable del sistema. Su contenido es el CODE de la última tecla pulsada.

Verdaderamente no hace mucho más de lo que hace con el INKEY$ excepto que se podría usar para escribir directamente un carácter. Si Vd. prueba este programa, verá que si pulsa una tecla, la tecla se indica en la pantalla en un breve momento a pesar de que el programa puede que no tenga más de la línea 50 cuando Vd. pulsó la tecla. El CODE de la última tecla pulsada queda almacenado aquí y permanece aquí hasta que no se pulsa otra tecla. Es posible verificar una nueva tecla pulsada examinando el bit 5 de la variable del sistema FLAGS 23611. Sería 1 para una tecla pulsada.

```
  10 PRINT "PRESIONE UNA TECLA A
HORA"
  20 FOR A=1 TO 900
  30 NEXT A
  40 CLS
  50 LET A=PEEK 23560
  60 PRINT A: IF A>31 THEN PRINT
 CHR$ A
```

Esto podría utilizarse para verificar una situación de tipo s/n (sí o no) – si Vd. sabía que iba a aparecer una podría indicar su respuesta antes de que el programa llegara allí y el programa respondería cuando así lo hiciera. También, si dos teclas fueron pulsadas simultáneamente el programa respondería si una se dejara de pulsar sin tener que esperar el teclado que la dejara completamente.

Los caracteres de control pueden generarse usando CAPS SHIFT en conjunción con las teclas de números. El ENTER retorna 13. Pulsando ambas teclas SHIFT a la vez retorna 14. Para ver esto, pruebe este programa:

```
  10 LET A=PEEK 23560
  20 PRINT A,CHR$ A AND A>31
  30 GO TO 10
```

23561 REPDEL Repetición de Espera

Esta variable del sistema contiene espacio de tiempo que una tecla debe mantenerse pulsada antes de que comience a autorepetirse. El tiempo de espera en Europa es de una quinceava parte de un segundo y comienza en 35/50 de un segundo. Por ejemplo, Vd. puede perfectamente POKEar esto si Vd. quiere que la tecla comience la repetición inmediatamente. Los cursores son más difíciles para controlar si Vd. dice POKE 23561,1. Efectivamente, el POKE 23561,0 desactiva la auto–repetición, dando un tiempo de espera de alrededor de unos 5 segundos como el POKE 23561,255.

23562 REPPER tiempo de espera entre repeticiones

Esta variable del sistema controla el espacio de tiempo entre las repeticiones una vez que el sistema de autorepetición ha comenzado. En Europa el tiempo es de la quinceava parte de un segundo. Si Vd. verdaderamente quiere desactivar la autorepetición por cualquier razón, el POKE 23562,0 o el POKE 23562,255 da unos 5 segundos entre las repeticiones. Si Vd. quiere editar líneas de un programa muy largo (es decir, una larga sentencia PRINT) entonces el POKE 23652,1 moverá el cursor al lugar adecuado. Pero procure no cambiar el 23561 demasiadas veces al mismo tiempo porque puede perder el control del cursor. El valor normal es 5/50 de un segundo o una décima parte de un segundo.

23563/4 DEFADD

Dirección del argumento de una función definida por el usuario en un programa, o sea, si Vd. tenía DEF FN A(B) en la línea del programa, el valor en 23563/4 será la dirección de la letra B entre paréntesis en esa línea mientras se use sólo la función. La mejor forma de hacer un PEEK dentro del 23563/4 para mostrar esto es poner el PEEK como parte del FN para ser evaluado como allí es siempre 0 a menos que la función esté siendo evaluada. Así pues, la línea:

```
  10 PRINT PEEK 23563+256*PEEK 2
3564
```

siempre retornará un 0. Por otro lado:

```
  10 DEF FN A(B)=PEEK 23563+256*
PEEK 23564
  20 PRINT FN A(999)
```

retornará la dirección de la B en la línea 10. El 999 no es significante, tan sólo algo que da un valor a la B para evitar un error. En el caso de una función sin argumento:

```
  10 DEF FN A()=PEEK 23563+256*P
EEK 23564
  20 PRINT FN A()
```

imprimirá la dirección del símbolo del paréntesis que cierra).

23568 a 23605 STRMS

Los primeros catorce bytes en un Spectrum básico contienen las direcciones relacionadas a los canales y corrientes. Las corrientes –3 a la +3 se almacenan en dos bytes cada una de ellas.

23606/7 CHARS

Esta variable del sistema tiene como valores normales:

23606 contiene 0

23607 contiene 60

Esta variable del sistema puntea al comienzo del conjunto de caracteres que el computador usa para imprimir en la pantalla y en la impresora. El SCREEN$ también usa esta variable del sistema. La dirección normal que se puntea es la 15360, la cual es 256 menos que la dirección del comienzo del conjunto de caracteres de la ROM. 256 menos porque el generador de caracteres se accede mediante algo similar al PEEK 23606 + 256 * PEEK 23607 + CODE" A " * 8 y ya que el primer carácter es ESPACIO, el CODIGO del ESPACIO es 32, y 8*32 es 256. El generador de caracteres tiene 768 bytes de longitud, de forma que si Vd. quiere fijar un nuevo conjunto de caracteres debe colocar este número de bytes en el caso de que sea sobregrabado por el BASIC –el sistema no se puede "colgar" pero lo que aparezca en la pantalla será totalmente ilegible–. Ya se había mencionado el SCREEN$ usando esta variable del sistema, de hecho, Vd. puede saber que el SCREEN$ generalmente no reconoce los gráficos definidos por el usuario, a menos que sean similares a un carácter existente del Spectrum. De hecho, el SCREEN$ funciona recogiendo las direcciones del comienzo del generador de caracteres observando la tabla hasta que encuentra un carácter de acoplo. Ahora como el Spectrum es un mapeado de bits en lugar de un mapa de la memoria como algunos computadores, todo lo que se imprima en la pantalla permanece igual aunque Vd. cambie los caracteres en la memoria. Así pues, temporalmente podríamos cambiar el puntero al conjunto de caracteres para puntear a los gráficos definidos por el usuario y observar allí. Un problema puede ser que a pesar de que hay una variable del sistema que nos dice donde comienzan los gráficos definidos por el usuario, esta dirección no es 256 menos. Así que debemos restar 256. Esto significa que restamos uno del byte más alto. Este programa se lo mostrará:

```
10 FOR X=144 TO 164
20 PRINT AT 0,0;CHR$ X
30 POKE 23606,PEEK 23675
40 POKE 23607,PEEK 23676-1
50 PRINT AT 20,0;SCREEN$ (0,0)
60 PAUSE 40
70 POKE 23606,0
80 POKE 23607,60
90 NEXT X
```

Lo que hicimos es hacer pensar al computador que los gráficos definidos por el usuario era el conjunto de caracteres normales. El SCREEN$ todavía produci-

rá caracteres con el CODIGO del 32 al 127. Ya que el SCREEN$ comienza con el CHR$32 y los UDGs comienzan en el 144, necesitaríamos añadir 112 para retornar los caracteres en el rango de los gráficos definidos por el usuario. Aquí hay una forma de hacerlo. La X es la coordenada x que va a través de la pantalla y la Y es la coordenada y que va a lo largo de la pantalla de la posición que el SCREEN$ debe examinar. Si allí el SCREEN$ no encuentra ninguno de los caracteres normales, entonces retorna si se encuentra uno. El carácter en Y,X aparece como A$. La línea 8025 sólo se necesita si está usando un conjunto de caracteres que no sea de la ROM. Si Vd. está usando el conjunto de caracteres de la ROM, entonces borre la línea 8025 y reemplace las líneas 8070 y 8080 con las versiones alternativas que vienen a continuación.

```
8000 REM PANTALLA PARA CREAR GRA
FICOS
8010 LET A$=SCREEN$ (Y,X)
8020 IF A$<>"" THEN RETURN
8025 LET A=PEEK 23606: LET B=PEE
K 23607
8030 POKE 23606,PEEK 23675
8040 POKE 23607,PEEK 23676-1
8050 LET A$=SCREEN$ (Y,X)
8060 IF A$<>"" THEN LET A$=CHR$
(CODE A$+112)
8070 POKE 23606,A
8080 POKE 23607,B
8090 RETURN
```

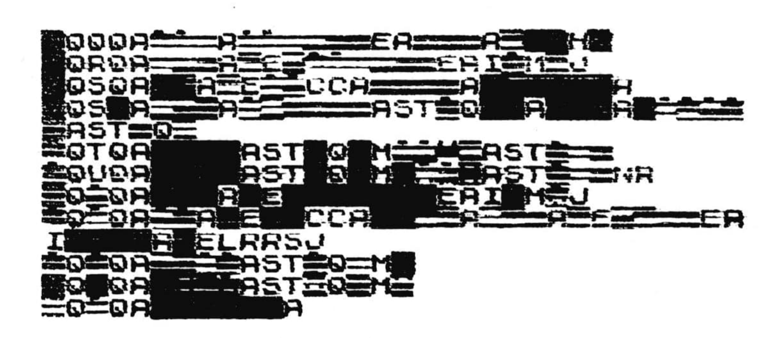
```
8070 POKE 23606,0
8080 POKE 23607,60
```

Pero la historia no termina aquí. Solamente hay 21 gráficos definidos por el usuario. Si SCREEN$ no encaja con ninguno de ellos, continuará buscando después de los gráficos definidos por el usuario hasta que haya terminado de buscar entre los rangos 32 al 127 que piensa que está buscando. Esto sería bastante embarazoso si solamente fueran algunos datos almacenados sobre los UDGs por la sola razón que reensamblara cualquier carácter. Para ayudarle a evitar esto, aunque los UGDs generalmente se encuentran en la parte superior de la RAM, se podría añadir esto:

```
6065 IF A$>CHR$ 164 THEN LET A$=
""
```

Incidentalmente, debería asegurarse de que 23606/7 siempre puntea al conjunto de caracteres correcto cuando haga un PRINT o un LIST, etc., esto es lo que se puede hacer para el correcto uso de la variable del sistema. Realmente es una

de las subrutinas de arriba sin la reinicialización del puntero CHARS. ¿Verdad que esto no se puede leer muy bien?.



23608 RASP

Controla la duración del sonido que suena, de forma que Vd. sepa que no le queda ya memoria. Cuando lo activa, tiene el valor de 64. Esto puede cambiarse, pero existe un pequeño problema. El POKE 23608,0 nos da un clic muy corto en lugar de un sonido, especialmente si Vd. odia esos sonidos que satíricamente le sacan de quicio cuando está ejecutando fuera de la memoria. Alternativamente, el POKE 23608,255 nos da un sonido muy largo que inmoviliza el teclado, evitando así que Vd. pueda escribir mientras dura el sonido.

23609 PIP

Controla la longitud del clic que se percibe cuando se pulsa una tecla en un modo de comando o durante un INPUT. Lo reemprende en el 0 pero puede cambiarse. Cualquier valor entre el 30 y el 130 nos da un sonido mucho más agradable y audible que el que percibimos con el clic. Los valores mayores de 130 ofrecen un sonido mucho más bajo, ya que computa los stops cuando suena el bip. Generalmente, el que se debe usar es el POKE 23609,100

23610 ERR NR

Controla el número del reporte de un error y generalmente tiene un valor de 255 a menos que aparezca un error, cuando contiene uno menos que los códigos del reporte del error, o sea, para el error 4, no hay memoria, contendría el 3. El mensaje que se imprime está contenido al principio de la ROM en la dirección decimal 5010. El final del mensaje se reconoce por el último carácter del mensaje que tiene el bit 7 fijado a 1. Después de los mensajes de error aparece © 1982 Sinclair Research Ltd., mensaje que deberá ver después de activar o ha-

cer un NEW. Vd. puede hacer un POKE 23610 para generar un error para detener el programa, pero como el mensaje es fijo y está en la ROM, puede acabar con un rebase de capacidad en la memoria. Si quisiera simular un error de no hay memoria terminaría el programa con un POKE 23610,3. Esto funcionaría como cualquier línea de programa, debería asegurarse de que fuera la última línea igual que aparece una condición para terminar un programa, sólo entonces el 23610 se trata como datos para determinar lo que está impreso.

23611 FLAGS (Señalizadores)

La variable de este sistema contiene varios flags que controlan el sistema del BASIC y generalmente no deben ser POKEados. Sin embargo, a algunos de estos flags se les puede hacer un PEEK.

BIT0: Siendo 1 indica que no se imprime ningún espacio antes de la siguiente palabra clave.

BIT1: Siendo 1 para indicar la impresión que se ha de enviar a la impresora. 0 significa enviarlo a la pantalla de TV.

BIT5: Cualquier tecla pulsada se indica por su CODE siendo almacenado en el 23560 (la variable del sistema LASTK) y el bit 5 del 23611 (FLAGS) se determina para indicar que se ha pulsado una nueva tecla.

BIT7: Sintaxis del flag.

Todo esto le será de más utilidad al programador del BASIC cuando use las rutinas de la ROM en un programa en código de máquina.

23613/4 ERR–SP

Guarda las pistas de la dirección de la pila de la máquina donde residen los datos que retornan adecuadamente. Intente llamar a unos cuantos GOSUBs sin acoplar ningún RETURN y vea lo que sucede en la memoria. Ahora Vd. puede ver lo que sucede y por qué cuando se encuentra sin memoria en una situación como ésta. Intente también POKEar el contenido de las tres direcciones, la base por la cual está punteado por el 23613/4, para ver en qué consisten los datos de retorno.

```
  10 LET A=PEEK 23613+256*PEEK 2
3614
  20 PRINT PEEK A;TAB 10;PEEK (A
+1);TAB 20;PEEK (A+2)
```

23617 MODE

Especifica el cursor. El 0.1,2. o 4 para el modo L/C, modo E, modo G o modo K respectivamente. El POKE de la variable de este sistema afectará la apariencia del cursor. Puede aparecer como una letra destellante, número, símbolo o incluso una palabra clave. Esto es más aparente durante una sentencia INPUT. El valor se reinicializa cuando se necesita, o sea, un cambio de modo desde el teclado. Pero si Vd. se encuentra en dificultades, pulse ambas teclas SHIFT para el modo E y después haga lo mismo para regresar al modo L/C. Pruebe este programa, el cual POKEa todos los valores posibles dentro del 23617. La mayoría son variantes de los cuatro cursores, es decir, se encontrará a Vd. mismo en un modo particular después del POKE, como por ejemplo, que todo aparece como gráficos así como en el modo G. El modo L/C, el 255, ofrecerá el símbolo destellante para puntearle donde está escribiendo...

```
10 FOR A=0 TO 255
20 PRINT A
30 POKE 23617,A
40 INPUT A$
50 NEXT A
```

23618/9 NEWPPC y 23620 NSPPC

El 23618 es la variable de un sistema de dos bytes que contiene el número de línea a donde tiene que dirigirse. El 23618 contiene el byte más inferior del número de línea y el 23619 el más alto, de forma que el número de línea se lee como PEEK23618+256K23619. Para POKEar un número de línea, por ejemplo, la línea X:

POKE23618, X – 256 * INT (X/256)

POKE23619,INT (X/256)

Ahora pasamos a ver la variable de sistema 23620. Con la 23618/9 y la 23620 podríamos realmente simular un GOTO a una sentencia con una línea de programa que nunca sería necesaria. Los GOTOs no pueden acceder a sentencias individuales con líneas de programa largas.

Para saltar a la línea 4 de una sentencia X, primero pasemos por los apartados arriba mencionados, entonces haga un POKE 23620,4 y se ejecuta el salto.

23624 BORDCR

Los bits de la variable del sistema controlan los atributos de la pantalla inferior y el color del BORDER de la forma siguiente:

bit	7	6	5	4	3	2	1	Ø
	FLASH de la parte inferior de la pantalla	BRIGHT de la parte inferior de la pantalla	Color BORDER y PAPER parte inferior de la pantalla			Color INK de la parte inferior de la pantalla		

Haciendo varios POKEs de los valores de la variable dentro de este sistema podría efectuar una multicolor y destellante pantalla inferior, o hacer que tanto el PAPER como el INK sean del mismo color para evitar que otra gente pueda introducirse en sus programas. Cualquier alteración debería hacerse ciegamente. O podría hacer que los INPUTs resultaran muy brillantes para resaltar.

23629/30 DEST

La dirección de la variable cuando es asignada. Si la variable se ha fijado antes, punteará al comienzo donde fue almacenada en el área de las variables. Si fuera el caso de que se definiera por primera vez, puntearía a la dirección del comienzo del nombre de la variable en el programa, es decir, en 10 LET A = 5 puntearía a la dirección de la letra A. También puede utilizarse para encontrar la dirección de la memoria de una variable numérica, si Vd. usa algo así como LET A = A :

```
  10 LET A=5
  20 LET A=A
  30 PRINT PEEK 23629+256*PEEK 2
3630
```

23631/2 CHANS

Almacena la dirección de donde comienza el área de información del canal.

23633/4 CHURCHL

Dirección de la información INPUT/OUTPUT usada en ese momento. Generalmente puntea durante la operación de un INPUT/OUTPUT a un bloque de 5 bytes en el área de información del canal. Use esto para examinar los contenidos.

```
1 FOR X=0 TO 3: PRINT #X;PEEK
 23633+256*PEEK 23634: NEXT X: P
AUSE 0: STOP
```

23627/8 VARS

Puntea al comienzo del almacén de las variables. Aparte de encontrarle el camino dentro del área de las variables, puede encontrar la longitud del programa BASIC con esta expresión. Esto incluye la pantalla, las variables del sistema, stacks y variables.

```
LET BYTES=PEEK 23627+256*PEEK 23
628-PEEK 23635-256*PEEK 23636
```

23635/6 PROG

Dirección del comienzo del área en la memoria donde el programa BASIC está almacenado. Puntea el primer byte del número de línea de la primera línea del programa. Puede ser útil si está convirtiendo los programas para otros computadores Sinclair con la información contenida como una sentencia REM en la primera línea de un programa. Véalo arriba también bajo el VARS.

Si quiere una línea de "seguridad" en un programa, mediante esta variable de sistema puede POKEar un cero en ambos bytes del número de la línea al comienzo de un programa. Las líneas del programa empiezan con un número de línea de dos bytes.

23637/8 NXTLIN

Dirección del comienzo de la siguiente línea de programa. Podría utilizar ésta para poder acceder al código de máquina almacenado junto con las sentencias REM en cualquier parte del programa, o sea, las que están grabadas con el MERGE desde una librería de subrutinas de una cinta. Tendrían sus propias llamadas para el código de máquina como esto:

```
9000 LET A=USR (PEEK 23637+256*P
EEK 23638+5)
9010 REM CODIGO MAQUINA
9020 RETURN
```

Una nota a añadir a esto es que no debería incluir ningún color, flash, brillo, etc., caracteres de control dentro de una sentencia REM o podrían ser inter-

pretados como códigos de máquina. Sin embargo, si proceden de una librería de subrutinas, generalmente esto no se utilizaría.

23639/40 DATADD

Contiene la dirección de la coma que termina el último elemento de DATA. Si nada se leyera de la lista (o sea, después de un RUN, etc.). la dirección contenida en la 23639/40 es la dirección de los bytes antes del área del programa, normalmente el CHR$ 128 al final del área de información del canal. Para demostrarlo ejecute este programa:

```
  10 DATA "1","2","3","4","5"
  20 LET A=PEEK 23639+256*PEEK 2
3640
  30 PRINT A;TAB 9;PEEK A;TAB 18
;CHR$ PEEK A AND PEEK A>31
  40 READ B$
  50 GO TO 20

23754     128
23763     44        ,
23767     44        ,
23771     44        ,
23775     44        ,
23779     13
```

La dirección en estas variables de sistemas de dos bytes pueden puntear al carácter ENTER o al punto y coma indicando el final de la línea o la sentencia que contiene los DATA. La dirección del terminador del último elemento de datos.

23641/2 E LINE

Esta variable de sistema puntea el comienzo del área sobre las variables. De aquí podemos tener una idea de cuánta memoria se usa en bytes para la pantalla, variables de sistema, programa y variables una vez el programa ha sido ejecutado para determinar las variables, etc. Escriba lo siguiente como un comando directo:

```
PRINT PEEK 23641+256*PEEK 23642-
16384
```

También podemos saber cuánto espacio se usa para las variables una vez se ha ejecutado el programa para fijar las variables. Use el comando:

```
PRINT PEEK 23641+256*PEEK 23642-
PEEK 23627-256*PEEK 23628
```

23653/4 STKEND

Esta variable de sistema contiene la dirección donde comienza la memoria de reserva. Leyendo esto llegamos a la conclusión de cuánta memoria nos queda sustrayéndola del RAMTOP. Esto no incluirá la memoria ya utilizada para los stacks de la máquina/GOSUB, pero incluye la longitud de la sentencia PEEK. Así pues, esto solamente es una breve guía aunque está muy bien adecuada para la mayoría de las circunstancias.

```
PRINT PEEK 23730+256*PEEK 23731-
PEEK 23653-256*PEEK 23654
```

23658 FLAGS 2

Esta variable de sistema contiene varios flags usados (generalmente) por el computador para indicar ciertas condiciones.

El mejor uso que podemos hacer de esto es hacer uso del flag indicado por el bit 3. El 1 indica si está activado o no el CAPS LOCK.

¿Qué uso es éste?. Consideremos en un programa el uso del INKEY$, es decir, en un menú de opciones en un programa para llenar, a menudo necesitamos saber si el operador está pulsando cierta tecla. Si al operador se le ha invitado a contestar Y para Sí o N para No él o ella debe pulsar y para Sí o n para No – mezcla las letras minúsculas con las mayúsculas. En la mayoría de los casos esto depende de si el CAPS LOCK está o no activado; pero a la mayoría de la gente no le importa si está en mayúsculas o minúsculas– cuando pulsan y o Y esperan que el computador comprenda que es lo mismo la y que la Y. Pero el computador no lo llega a apreciar. Así pues, si activamos el CAPS LOCK automáticamente, no debemos preocuparnos más ya que tenemos un programa el cual no tiene que revisar (siempre que le concierna) las dos opciones separadas para cada opción.

Resulta tentador usar la sentencia en BASIC POKE23658,8 para activar el CAPS LOCK y la POKE23658,0 para desactivarlo. Pero esto afectará a los otros flags, así que primero verifique su primer estado a menos que sepa que no tienen ningún valor en particular. Normalmente en el modo L, la 23658 tiene un valor de 0, así que generalmente está bien usar los POKEs de arriba. El resulta-

do no es que vaya a romper algo,pero sí que obtendrá unos efectos muy divertidos. Cuando el buffer de la impresora esté vacío el bit 1 será cero.

23659 DF SZ

Esta variable de sistema contiene el número de líneas en la sección inferior de la pantalla, generalmente utilizada por los INPUTs, reportes de error, etc. Normalmente sería un 2, excepto cuando en la pantalla aparece un mensaje INPUT demasiado largo, etc. Si se hace un POKE de un valor de 0 para intentar borrar esa parte de la pantalla,que no se usa con tanta frecuencia,de manera que podemos usar las 24 líneas de la pantalla, el computador se queda "colgado". Sin embargo se puede hacer con algunas restricciones. Estas son que debemos asegurarnos que la parte inferior de la pantalla vuelve a su estado normal antes de utilizarla para algo como esto – así que ROMPER un programa será algo así como ¡catastrófico!. También los errores que se han generado a lo largo de un programa tendrán el mismo efecto ya que el reporte del error se imprimirá en la pantalla. A continuación viene un breve listado para demostrar el uso de la línea 22 y 23 en la pantalla. Desafortunadamente, sólo funciona con las instrucciones PRINT o PRINT TAB ya que no podemos usar el PLOT aquí y el PRINT AT sólo funciona en la línea 22. La pantalla vuelve a su estado normal con el POKE23659,2 junto con el programa.

```
10 POKE 23659,0
20 FOR A=0 TO 23
30 PRINT A
40 NEXT A
50 PAUSE 0
60 POKE 23659,2
```

Para demostrar lo que puede ir mal generaremos un error añadiendo esta línea al programa:

```
45 PRINT ERROR
```

¡Caramba! Si Vd. quiere imprimir en las dos líneas inferiores es mucho mejor usar el PRINT#1;"texto" que funciona si no igual de bien mucho mejor,sin el riesgo de dejar "colgado" el sistema. Si Vd. hace un POKE de un valor más alto que 2 dentro del DF SZ la pantalla superior se convertirá en más pequeña de lo normal. Así pues,después de un POKE23569,Y la pantalla superior será de 24 hileras Y, y se deslizará cuando la posición PRINT alcance o sobrepase las 24 hileras Y,0. Este programa muestra como puede mantenerse el deslizamiento

de una parte de la pantalla con el DF SZ y el SCR CT. Aquí aparecen números aleatorios y sólo deslizan hacia arriba 14 líneas.

```
10 POKE 23692,0: POKE 23659,10
20 PRINT RND
30 GO TO 10
```

23670/1 SEED

Cuando se usa la función RANDOM (número), el número (una constante o una variable) se almacena en esta variable de sistema. Este es el número que determina el siguiente número aleatorio. Abre la posibilidad de engañarle ya que Vd. trabajaría sobre el siguiente (presuntamente) número aleatorio generado y usaría el conocimiento obtenido "balanceando" la suerte a su favor. Por ejemplo, después de hacer un RANDOM 1 el siguiente valor del RND sería 0.0022735596 o INT (RND*6) + 1 para simular al tirar un dado sería 1.

23672/3/4 FRAMES

Es un contador de cuadros que puede utilizarse como temporizador. Cuenta los cuadros de una TV, de forma que en el Reino Unido se incrementa cincuenta veces por segundo, o cada 0.02 segundos, aunque el tiempo que tarda en leer y evaluar estos tres bytes del temporizador puede que no le permita ser utilizado con esta precisión. Tiene un rango temporizador de unos 4 días (alrededor de unos 3 días y 21 3/4 horas). El manual (capítulo 18) precisa que Vd. necesita leer el valor de estos tres bytes dos veces y sucesivamente tomar el valor más alto para mayor precisión debido a la posibilidad de que los valores de los tres cambien mientras se están leyendo y causen grandes imprecisiones.

Debe precisarse también que los bytes del temporizador están en orden opuesto al que Vd. esperaba. El byte más significante es el 23674, de forma que los valores del temporizador están listos:

```
65536*PEEK 23674+256*PEEK 23673+
PEEK 23672
```

el cual retorna en unidades de cincuenta de un segundo. Existen varias cosas que afectan la precisión de este temporizador. Usando el BEEP detiene el temporizador. Usando la impresora y cargar/salvar, etc., también afectan a su pre-

cisión. Sin embargo, el uso de PAUSE está bien ya que sólo espera un determinado tiempo sin reinicializar o detener el temporizador.

23675/6 UDG

La dirección del comienzo de los patrones de puntos para los gráficos definidos por el usuario es generalmente 32600 en un Spectrum de 16K o de 65368 en un Spectrum de 48K. Este número es el mismo que USR"a", de forma que imprimir USR"a" corresponde a:

PRINT PEEK23675 + 256 * PEEK23676

Los "POKEadores compulsivos" pueden divertirse bastante con éste. El manual sugiere cambiar esto para tener más espacio teniendo menos gráficos definidos por el usuario. Sin embargo, también es posible hacer lo contrario, determinar más de un gráfico definido por el usuario aunque un solo conjunto de 21 puede estar en uso a la vez. Recuerde que desde que hay 21 UDGs es necesario determinar aparte 21*8 o 168 bytes para cada conjunto separado de UDGs y hacer un POKE del comienzo de las direcciones, dentro de la 23675/6, del conjunto de caracteres en uso.

Para divertirse, escriba los siguientes comandos:

POKE23675,96: POKE23676,127 (Spectrum de 16K)

POKE23675,96: POKE23676,255 (Spectrum de 48K)

Entonces usando los gráficos definidos por el usuario (normalmente aparecen en mayúsculas hasta que son redefinidos) intente escribir un mensaje. Le dejo para que descubra Vd. solo lo que sucede...

Un consejo útil: una vez ha determinado el conjunto de caracteres definidos por el usuario puede hacer un SAVE en la cinta. La mayoría de la gente escribiría algo así:

SAVE "caracteres"CODE32600,168

Bien, pero Vd. tiene que especificar el comienzo de las direcciones. Vd. podría usar SAVE "caracteres" CODE(PEEK23675+256*PEEK23676), 168 y salvar el actual conjunto de UDGs en la cinta sin saber dónde comienzan las direcciones. Por ejemplo, esto le permitiría hacer un LOAD del conjunto de caracteres salvados desde un Spectrum de 16K a un Spectrum de 48K sin conocer las direcciones. Para obtener otra vez el conjunto de caracteres en su lugar correcto

en una máquina con diferentes cantidades de memoria simplemente use LOAD "caracteres" CODE(PEEK 23675 + 256*PEEK23676),168. Esto haría relocalizar los datos automáticamente en la dirección correcta para la máquina que se usa en ese momento. Esto es lo mismo que LOAD "caracteres" CODE USR"a" que salva un bit de escritura aunque pueda parecer un bit algo extraño.

23679 P POSN

Contiene información de hasta donde abarca el buffer de impresión del LPRINT de la impresora. Contiene (33 – número de la columna) para las columnas de la 0 a la 31. Vd. no puede cambiar la posición LPRINT haciendo un POKE de esto sólo.

23680 PRCC

Contiene el byte inferior de la dirección donde va a ir el siguiente carácter dentro del buffer de la impresora, o sea, contendrá (23296 + LPRINT número de la columna), siendo 0 para la columna izquierda de la impresora, 15 para la columna 15ava, etc. Como esto es la dirección de la hilera superior de puntos de cada carácter al que Vd. puede hacer un POKE para cambiar la posición del buffer LPRINT, cambie el valor en P POSN (23679) para el acoplamiento. Incluso puede parecer que funciona si Vd. no hace esto pero se encontrará con problemas al final de la línea.

23681 VARIABLE DE SISTEMA NO USADA

Esta variable de sistema, aunque estrictamente hablando no se usa, generalmente contiene 91. Este es el byte más alto de la dirección del buffer LPRINT (91 * 256 es 23296 donde comienza el buffer). Puede hacer un POKE para su propio uso pero si usa la impresora lo volverá a escribir otra vez a 91.23680/1 que junto contiene la dirección de la posición LPRINT en el buffer de la impresora. No causará ningún efecto en la impresora si Vd. hace un POKE 23681 pero nada de lo que hay aquí almacenado puede ser sobreescrito por las rutinas de la impresora.

23677/8 COORDS

23677 es la variable del sistema que contiene las coordenadas X del último punto dibujado. Después del CLS se reemprende en el 0 y el 23678 es la variable del sistema que contiene la coordenada Y del último punto dibujado. Contiene el valor actual, de forma que si el último punto dibujado era 3,3 ambos bytes contendrán 3.

Estos dos pueden ser POKEados con las coordenadas válidas X e Y respectivamente. Como este POKE no dibuja nada en la pantalla, es una forma conveniente de mover el cursor PLOT. Podría hacerse con el PLOT OVER 1;X,Y:PLOT OVER1;X,Y pero el resultado sería nefasto. Entre otras cosas simularía el movimiento que se puede encontrar en cualquier otro BASIC. Es útil si quiere dibujar líneas desde un determinado punto.

23684/5 DF CC

Dirección en el fichero de visualización de la posición PRINT. Podría utilizarse un POKE para enviar el resultado del PRINT en algún otro lugar pero para ello necesita conocer más la forma en que se organiza un fichero.

23688/9 SPOSN

La 23688 contiene la información concerniente a saber cuán lejos ha llegado la posición PRINT en la pantalla. Se reemprende como 33 para el lado izquierdo de la pantalla y desciende uno por uno cada vez que la posición PRINT se mueve un lugar a la derecha. Después de usar PRINT AT Y,X; (si Y y X son coordenadas de PRINT AT válidas) el 23688 contendría 33 – X. Esto sería útil cuando intentara evitar que las palabras quedaran cortadas por la mitad cuando se imprimieran en la pantalla. Si Vd. se imagina el número 23688 descontándolo a cero y no hubiera más espacio en la línea, puede ver esto comparado con la longitud de la palabra a imprimirse nos daría una idea de cuándo sería necesario movernos a una nueva línea para evitar cortar las palabras. Supongamos que la palabra a imprimir fuera W$:

```
IF PEEK23688<LEN W$+1 THEN PRINT
```

Esto sólo funciona para las palabras con menos de 32 caracteres de longitud. El 23689 contiene la información relacionada con la distancia que ha de llegar la posición PRINT a lo largo de la pantalla. Se reemprende en la línea 24 de la parte superior de la pantalla y desciende uno por uno cada vez que la posición PRINT se mueve un carácter hacia abajo. Si no quiere que la visualización en la pantalla se deslice o que se borrase cuando la posición PRINT alcanzara la parte inferior de la pantalla, pruebe esto:

```
IF PEEK 23689=3 THEN CLS
```

23692 SCR CT

Contiene cuantos deslizamientos se llevarán a cabo + 1 antes de esperar el deslizamiento para poder visualizar la pantalla. En los juegos de gráficos, espe-

cialmente, no tiene ningún sentido ya que uno no está interesado en esperar para ver. Así pues, si el número 23692 es todo menos un 1 la espera no sucede. Entonces, ¿el POKE23692,255 nos dará las 255 líneas de impresión antes de que la máquina espera con el deslizamiento?. El POKE 23692,0 parece tener un efecto similar excepto que Vd. tiene una línea de más impresión. Si necesita más, y la impresión se hace con un bucle será necesario incluir también la sentencia POKE en el bucle. Se pierde tiempo pero es necesario.

23693 ATTR P

Contiene atributos permanentes, o los atributos en efecto globalmente (FLASH, BRIGHT, PAPER, INK). Los colores locales en las sentencias PRINT, etc., tienen que ver con algo más. Observe que la mayoría de las rutinas ROM usan los valores de la variable del sistema que manejan los atributos temporalmente a menos que se especifique un parámetro local. Sin embargo, el CLS borra la pantalla de los colores, etc., en el ATTR P. Las funciones de los bits individuales son las siguientes:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
FLASH	BRIGHT	Color PAPER en binario			Color INK en binario		

BIT 7 es 1 para FLASH 1.

BIT 7 es 0 para FLASH 0.

BIT 6 es 1 para BRIGHT 1.

BIT 6 es 0 para BRIGHT 0.

Los bits 5, 4 y 3 contienen el color del PAPER en binario, es decir, para

PAPER 7 bits 5, 4 y 3 sería 111.

Los bits 2, 1 y 0 contienen el color INK en binario, o sea, para

INK3, los bits 2, 1 y 0 sería 011.

Los atributos del 8 o 9 no tienen ninguna relación aquí. Si los atributos permanentes son el 8 o 9 entonces si los almacena en la 23693 no es válido.

23694 MASK P

Esta es la variable del sistema que ayuda al Spectrum a determinar los atributos de cualquier cosa impresa cuando se especifica un parámetro de 8. Así pues, si Vd. especificó BRIGHT 8 globalmente, el bit 6 de la 23694 se determinará a 1 para recordar al computador en el futuro que BRIGHT 8 ha sido especificado. Para determinar cuándo imprime el color, destello y brillo, el computador mira lo que ya tiene allí e imprime la palabra en ese color, etc. O si Vd. prefiere, solamente sobreimprime el carácter en la pantalla y deja los atributos solos. Esto es lo que hace cada bit:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
FLASH 8	BRIGHT 8	PAPER color 8			INK color 8 ?		

Bit 7 es 1 cuando FLASH 8 está en efecto.
Bit 6 es 1 cuando BRIGHT 8 está en efecto.
Los bits 5, 4 y 3 son normalmente todos 1 cuando PAPER 8 está en efecto, pero mire abajo.
Los bits 2, 1 y 0 son normalmente todos 1 cuando INK 8 está en efecto, pero mire abajo.

Cuando hay más de un bit a considerar, como el INK o PAPER, entonces solamente el conjunto de bits tienen sus atributos de bits tomados de la pantalla. Esto puede acarrearle algunos efectos inesperados. Pruebe esto:

```
10 INK 8
20 POKE 23694,BIN 00000011
30 PRINT AT 0,0; INK 5;"555555
5"
40 PRINT AT 0,0;"1111"
```

Como el INK 8 está especificado, Vd. espera que se impriman los unos en color como los cinco, pero no. En lugar de revisar los atributos INK como un todo, lo que hace es revisar sólo el conjunto de bits en la 23694, donde estaban los bits 0 y 1. Intente de adivinar en qué color se imprimirán los unos. ¡Diviértase un poco!.

23695 ATTR T

Esta variable del sistema contiene el color temporal tal como fue determinado por las sentencias locales junto con las sentencias PRINT.

Esto lo podría ver Vd. mismo con estos dos comandos directos:

PRINT PEEK23695

PRINT INK7;PAPER0;PEEK23695

O sea, incluye el PEEK en una sentencia PRINT bajo el efecto de los controles del color local. Normalmente, y a menos que las sentencias del color local sean especificadas, esta variable del sistema contendrá los valores del color global. Los colores, etc., que se van a usar para imprimir en la pantalla son tomados de estas variables temporales del sistema y las cosas quedan equilibradas de modo que el ATTR T sólo es diferente del ATTR P si hay atributos locales de colores. Esta es la función de los bits individuales:.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
FLASH temporal	BRIGHT temporal	Color PAPER temporal			Color INK temporal		

23696 MASK T

Esta es muy parecida al MASK P (variable del sistema 23694) excepto que los parámetros aquí son temporales. Normalmente sucede igual con los parámetros 8 permanentes, aunque esto cambia con los colores 8 locales, etc. están en efecto. Podría estudiar esto con algo así:

```
PRINT PEEK 23696,INK 8;PEEK 2369
6, INK 0;FLASH 8; PEEK 23696
```

Los bits individuales tienen las siguientes funciones:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
FLASH 8 temporal	BRIGHT 8 temporal	PAPER 8 temporal			INK 8 temporal		

23697 P FLAG

Esta variable del sistema contiene, como Vd. puede adivinar por el nombre, flags usados durante la impresión. Después de haber especificado el PAPER 9, los bits 6 y 7 se fijan a 1. Después de haber especificado el INK 9, los bits 4 y 5 se fijan a 1. Después de haber especificado el INVERSE 1, los bits 2 y 3 se fijan a 1. Después de haber especificado el OVER 1, los bits 0 y 1 se fijan a 1. Los efectos son globales si los bits impares (bits 1, 3, 5 y 7) se fijan a 1 y tempo-

ralmente si los bits pares (bits 0, 2, 4 y 6) se fijan a 1, tal como muestra este diagrama:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
global PAPER 9	temporal PAPER 9	global INK 9	temporal INK 9	global INVERS 1	temporal INVERS 1	global OVER 1	temporal OVER 1

23681,23728/9

Estos tres bytes en la variable de sistema no son normalmente usados por el Spectrum; los puede usar como "variables de usuario" para utilizar en sus propios programas a los que acceder información. Son particularmente útiles en rutinas en código de máquina donde Vd. simplemente puede acceder a la información mediante la dirección más que buscando la variable en el área de éstas. La 23728/9 es para usar para interrupciones no enmascarables pero esto no sucede con el Spectrum.

23730/1 RAMTOP

Esta variable del sistema de dos bytes apunta al último byte de la RAM del área del sistema del BASIC. Observe que esta dirección no es el final de la memoria usada por el BASIC, en el sentido de que los gráficos definidos por el usuario normalmente están escondidos tras esta dirección. Si Vd. mueve el RAMTOP hacia arriba por encima del comienzo de los gráficos definidos por el usuario pueden éstos quedar sobregrabados, pero Vd. gana unos cuantos valiosos bytes que les serán útiles a los usuarios de la máquina de 16K.

Una cosa importante es que el NEW sólo opera mientras la dirección esté contenida en la 23730/1 de manera que Vd. pueda almacenar los datos sobre esto, lo cual puede pasarse entre los programas cargados dentro del computador. Lo mismo sucede si Vd. quiere preservar las rutinas en código de máquina, etc.

23732/3 P RAMT

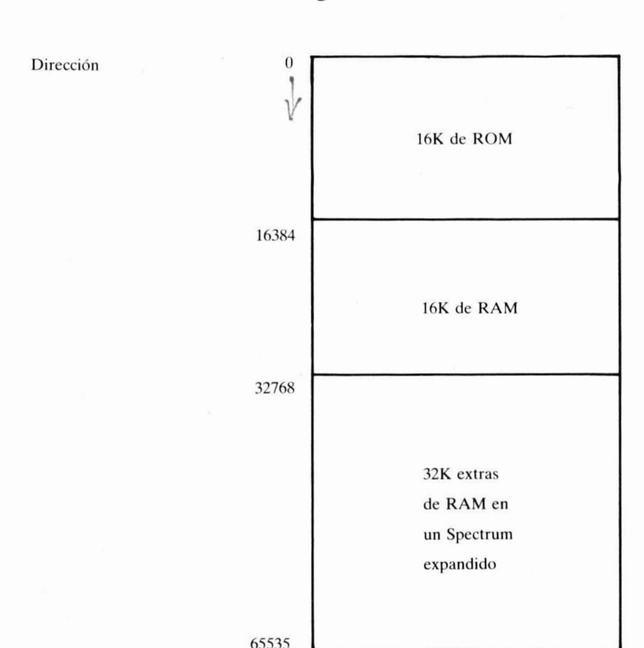
Esta contiene la dirección de donde la RAM termina en el Spectrum. Si Vd. posee un Spectrum cuya capacidad de memoria desconoce, entonces no necesita buscar en el interior para ver si es un modelo expandido o no, tan sólo entre la expresión:

PRINT PEEK23732+256*PEEK23733 – 16384

Los bytes 16384 sustraídos son para la ROM ya que la RAM comienza en la dirección 16384 y va hasta la dirección contenida en la P RAMT.

EL DISEÑO DE LA MEMORIA DEL SPECTRUM

Generalmente el Spectrum consiste de 16K de Memoria Sólo para Lectura (ROM) y de 16K de RAM (Memoria de Acceso Aleatorio) o 48K de RAM, como se muestra en el diagrama:



La ROM de 16K contiene las instrucciones, datos, etc., que el Spectrum necesita para ejecutar los programas, ejecutar las instrucciones suministradas por el usuario, manejar la impresora, grabador de cinta, etc. Esta área consiste de 16384 bytes (0 a 16383). Después de esto hay otros 16384 bytes de la RAM y otros 32768 bytes de la RAM en el Spectrum expandido a 48K.

La diferencia principal entre la ROM y la RAM es que los contenidos de la ROM son fijos y no pueden ser alterados y así permanecen aún cuando se desconecta la corriente. Los contenidos de la RAM pueden ser alterados bastante fácilmente, y naturalmente, los contenidos se pierden cuando no hay corriente. La RAM se divide en muchas secciones como muestra este diagrama:

Dirección	Area de Memoria	Cómo encontrar una dirección de frontera si no está fijada.
16384	Fichero de Pantalla	
22528	Atributos de Pantalla	
23296	Buffer de la Impresora	
23552	Variables del Sistema	
23734	Mapas del Microdrive (si está conectado)	
CHANS	Información del Canal	PEEK23631 + 256*PEEK23632
	CHR$128	
PROG	Programa BASIC	PEEK23635 + 256*PEEK23636
VARS	Area de almacenamiento de las Variables	PEEK23627 + 256*PEEK23628
	CHR$128	
	Comandos directos	
	CHR$ 128	
E LINE	Comando o línea modificado	PEEK23641 + 256*PEEK23642

	Comando o línea modificado	
	CHR$13	
	CHR$128	
WORKSP	Datos del INPUT	PEEK23649 + 256*PEEK23650
	CHR$13	
	Area de trabajo temporal a continuación del marcador CHR$ 13	
STKBOT	Stack del calculador	PEEK23651 + 256*PEEK23652
STKEND	Reserva de memoria	PEEK 23653 + 256*PEEK23654
Z80A Puntero del stack	Stack de la máquina usado por el microprocesador Z80A (no accesible desde el BASIC)	
	Stack de los GOSUBs	
RAMTOP	CHR$62	PEEK23730 + 256*PEEK23731
UDG	Gráficos definidos por el usuario	PEEK23675 +256*PEKK23676
P-RAMT		PEEK23732 +256*PEEK23733

Hagamos un breve repaso a estas secciones una por una donde sea de interés.

(1) Fichero de pantalla Dirección 16384 a la 22527
Es la copia del cuadro de la pantalla de la televisión que reside en la memoria del computador. Para cada punto en la pantalla tanto en la parte superior como en la parte inferior (256*192), existe un bit de acoplo en esta área. Consiste en 6144 bytes, los cuales tienen un diseño bastante curioso a primera vista tal como explica el manual. Si un punto en la pantalla se fija al color INK entonces el bit relevante se fija a 1 en el fichero de pantalla. Esta corto programa le dará una idea del diseño del fichero de pantalla.

```
10 FOR A=16384 TO 22527
20 POKE A,255
30 NEXT A
```

(2) Fichero de atributos Dirección 22528 a la 23295
Esta área de la memoria contiene información sobre los efectos de los colores, brillo y flashing en la pantalla. Si Vd. considera que el espacio de un carácter en la pantalla ocupa una casilla de 8 por 8 puntos entonces los puntos en esa casilla sólo tendrán un color INK y un color PAPER porque todos sus atributos vienen del mismo lugar. Los atributos operan en la misma rejilla como lo hace la rutina PRINT. Esta es la razón por la cual Vd. no puede tener un extraterrestre verde con los ojos de color magenta sobre un fondo negro u otro color. Un byte de ocho bits en el área de los atributos contiene la información para el FLASH, BRIGHT, PAPER, e INK para la posición de un carácter en la pantalla. Hay 768 bytes de atributos (32*34) incluidas las dos líneas inferiores de la pantalla utilizadas para los INPUTs, etc. Estos bytes de atributos se almacenan en el siguiente orden: la primera línea de 32 bytes para la primera línea de 32 caracteres de la pantalla, la segunda línea de 32 bytes de atributos para la segunda línea de 32 caracteres a través de la pantalla, y así sucesivamente. Este corto programa se lo mostrará:

```
10 FOR A=22528 TO 23295
20 POKE A,199
30 NEXT A
```

Se usan tres bits para PAPER, tres bits para INK, un bit para FLASH, y un bit para BRIGHT. Todos son almacenados en binario, así pues, un solo bit da 0 o 1 y tres bits dan 0 a 7. Debe tener en cuenta que los parámetros 8 o 9 (o sea

FLASH8 o PAPER9) no son almacenados aquí; solamente el resultado después de que el computador ha decidido si el INK9 será rojo o blanco. Esto es lo que cada bit significa en un byte de atributo:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit Ø
FLASH	BRIGHT	color PAPER			color INK		

(3) Buffer de la impresora Dirección 23296 a la 23551
Información sobre los caracteres que esperan ser enviados a la impresora. Consiste en 256 bytes o 32 por 8, de forma que los gráficos puedan ser enviados a la impresora. El actual patrón de puntos que aparecerá en el papel de la impresora está almacenado aquí.

(4) Las variables del sistema Dirección 23552 a la 23734
Bytes conteniendo la información relacionada a varios factores sobre la operación del computador, donde varias secciones de la memoria como los gráficos definidos por el usuario, variables, etc. comienzan y terminan. Todos ellos tienen un nombre, pero el computador no los reconocerá, ya que es sólo para el beneficio de los programadores. Estos nombres son SCR-CT, VARS, etc.

(5) Maps de los microdrives
Esta área contiene la información relacionada con los microdrives cuando están conectados. Si los microdrives no están conectados, no hay nada allí y el puntero de la siguiente variable del sistema CHANS apunta al 23734. En el momento de escribir este libro no había más información disponible.

(6) Canal de información
En un Spectrun estandar de 16K o 48K sin ningún otro accesorio (excepto una impresora, la cual no afecta para nada a los contenidos de esta área), hay detalles de 4 canales INPUT/OUTPUT en esta área. Estos se refieren a la información que se recibe y envía, es decir, cualquier cosa escrita en el teclado aparece en la parte inferior de la pantalla, y las direcciones de las rutinas que manejan las operaciones INPUT/OUTPUT para ese canal.

El área comienza en la dirección almacenada en la variable del sistema CHANS23631/2 y termina en la dirección contenida en la variable del sistema PROG23635/6 menos una con un byte que contiene el CHR$ 128 que significa el final del área de información del canal. En términos de bytes en el Spectrum de 16K o de 48K (sin importar si está o no conectado a la impresora) el área es de 21 bytes de largo desde la 23734 a la 23754. Los números contenidos en esta área son los siguientes. Ellos residen en una tabla de 4 por 5 figuras:

```
244   9   246   16   75   [K]
244   9   246   21   83   [S]
129  15   131   21   82   [R]
244   9   246   21   80   [P]
128
```

Los caracteres de la derecha son los "nombres del fichero" de los cuatro canales, así pues, tenemos el canal K, canal S, canal R y canal P.

El canal K es el canal del "teclado". La información llega del teclado y se dirige a la parte inferior de la pantalla. El canal R permite que la información sea enviada al área del espacio de trabajo y el canal P permite que la información sea enviada a la impresora. Así pues, ¿cuáles son los cuatro bytes anteriores a los nombres de los ficheros?. Reescribamos esto como dos direcciones y los nombres del fichero, las direcciones siendo las de las rutinas que contienen las rutinas apropiadas:

dirección de la rutina entrada (decimal)	**dirección de la rutina de salida (decimal)**	**nombres de los ficheros**
2548	4264	K
2548	5572	S
3969	5572	R
2548	5572	P

(7) Area del programa
Esta es el área donde va el programa en BASIC. La variable del sistema PROG apunta al comienzo del área del programa, al primer byte del primer número de línea. Esto sigue a un byte con el CHR$ 128 marcando el final del canal del área de información. Esta área termina en la dirección contenida en VARS menos 1. Las líneas de un programa en BASIC son almacenadas de la siguiente forma:

Dos bytes conteniendo el número de línea MSB | LSB	Dos bytes conteniendo la la longitud del texto de la línea más 1 para el CHR$ 13 al final de la línea	Texto de la línea de programa	CHR$ 13 BIN 00001101 (ENTER)

Cada línea termina con un CHR$ 13 o el carácter ENTER. Las sentencias múltiples están separadas por un punto y coma, el CHR$ 58, y los números son almacenados dos veces, primero como los códigos de sus dígitos, entonces, después de un CHR$ 14 (indicando un número) sigue la representación de un quinto byte del número ya sea un formato entero o un punto flotante.

Este simple programa le permitirá examinar cualquier línea del BASIC que Vd. coloque como primera línea de un programa. El mismo piensa dónde comienza la primera línea del programa (línea 10) y dónde termina (línea 20) usando los bytes tres y cuatro en la línea del programa, la cual contenía la longitud de la línea desde el byte 5 de la línea del programa hasta el carácter ENTER al final. A todas las direcciones se les hace un PEEK y se imprimen tres columnas en la pantalla. La primera muestra las direcciones, la segunda el número en esa posición de la memoria y la tercera el carácter correspondiente a ese número si es un carácter que se puede imprimir. El ejemplo muestra una sentencia REM siendo examinada:

```
23755    0
23756    1
23757    15
23758    0
23759    234      REM
23760    108      l
23761    105      i
23762    110      n
23763    101      e
23764    32
23765    111      o
23766    102      f
23767    32
23768    66       B
23769    65       A
23770    83       S
23771    73       I
23772    67       C
23773    13
```

```
   1 REM LINEA DE BASIC
  10 LET SALIDA=PEEK 23635+256*P
EEK 23635
  20 LET FINAL=SALIDA+3+PEEK (SA
LIDA+2)+256*PEEK (SALIDA+3)
  30 FOR A=SALIDA TO FINAL
```

```
40 LET ASIENTO=PEEK A
 50 PRINT A;TAB 8;ASIENTO;TAB 1
6;CHR$ (ASIENTO) AND ASIENTO>31
 60 NEXT A
```

(8) Area de las variables

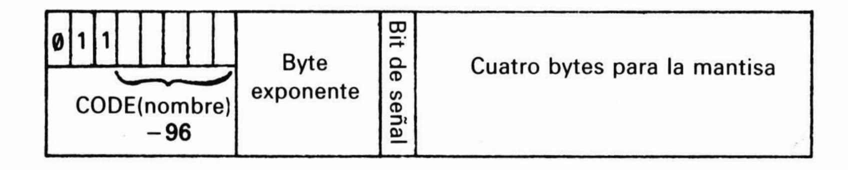
El área de las variables comienza en la dirección contenida en la variable del sistema VARS y termina en la dirección contenida en la variable del sistema E–LINE. Esto es donde las variables usadas por el BASIC son almacenadas. Para evitar confusiones, los nombres de las variables son almacenados con formatos diferentes a sus nombres con nombres de cadenas, matrices, variables numéricas, etc.

Diferentes patrones de bits para la primera letra en el nombre de la variable permiten al computador decir las diferencias entre los diferentes tipos.

Generalmente estas letras son almacenadas como letras en minúsculas, pero los patrones de bits diferentes indican que nosotros no siempre podemos buscar el CODE (CODIGO) de la letra en el nombre, por ejemplo. Ahora daremos un vistazo a los diferentes tipos de variables. La primera letra de todos los nombres de variables es almacenada en un byte como el código de la letra menos 96, ya que sólo los bits del 0 al 4 son significantes para el nombre. Los bits 5, 6 y 7 siempre tienen valores que dependen del tipo de variables, de forma que no se usan en el nombre. Los valores, si el bit 5 y 6 fueran 1, sería 32 y 64 respectivamente, la suma de lo cual es 96. La letra actual se almacena en los bits 0 al 4 como el código de la letra en minúscula menos el valor de estos dos bits. Así pues, si la primera letra del nombre de la variable era una "a", los bits del 4 al 0 serían BIN 00001. Los bits 5, 6 y 7 dependerían del tipo de variable.

Variables cuyo nombre consiste de una letra (variable numérica)

Este tipo de variable se almacena con los bits 5 y 6 de su nombre fijado a 1 y el bit 7 se reinicializa a 0. Este es un ejemplo raro del primer carácter en el nombre siendo almacenado exactamente igual a su nombre, pero sólo porque coinciden los patrones del bit. El nombre va seguido por el byte exponente y cuatro bytes para la mantisa:



El byte exponente tiene un valor del 1 al 255 (puede ser 1 o 255). Esto muestra donde reside el punto decimal. La mantisa está contenida en cuatro bytes. Estos dan los dígitos del número y pueden tener un valor de 0.5 a 1 (nunca es 1, aunque puede ser 0.5). El número puede encontrarse con la fórmula (mantisa)*2 (exponente–128).

Como resultado de los valores que la mantisa puede tomar, el bit del extremo izquierdo de la mantisa se usa como un bit de señal. Es 1 si el número es negativo o 0 si el número es positivo.

Hay un método un poco diferente de almacenar números completos desde el –65535 al +65535, números que pueden ser acomodados en dos bytes:

byte1	byte2	byte3	byte4	byte5
Ø Cero	Ø si es número positivo o 255 si es negativo	LSB	MSB	Ø Cero
		El número está contenido en el tercer y cuarto bytes		

El primer byte de números contenidos de esta forma siempre es cero. Este podría añadir identificación de este formato de números. El segundo byte es el de signo – será 0 si el número es positivo o será 255 si el número es negativo. El tercer y cuarto byte contiene el valor de todo el número en el orden del Byte Menos Significante (LSB) seguido por el Byte Más Significante (MSB) y está contenido en lo que llamamos "formato de complementos a dos".

El valor para un número positivo puede leerse mediante la expresión:

LSB+256*MSB

El valor para un número negativo puede encontrarse mediante la expresión:

LSB+256*MSB–65536

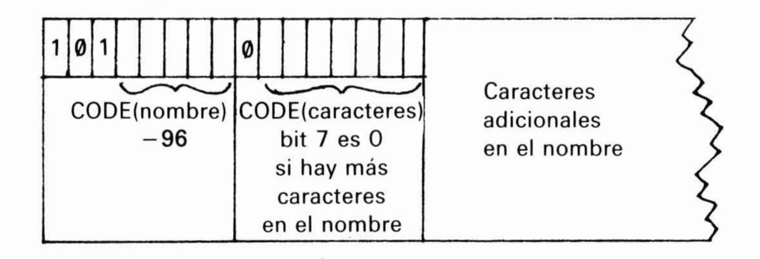
Y si alguna vez desea leer el valor de esta forma del número, Vd. podría usar:

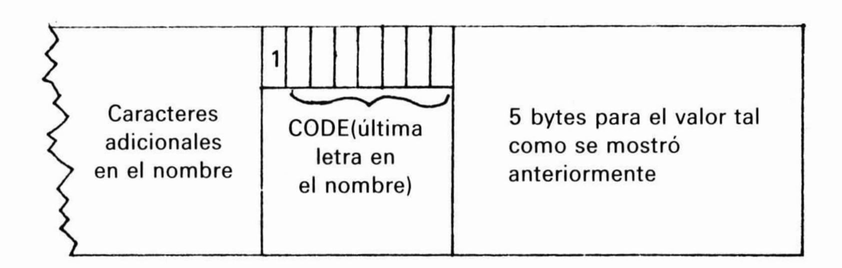
LET valor=LSB+256*MSB–(65536 AND(byte de señal=255))

El quinto byte siempre será cero.

Variables numéricas cuyos nombres consisten de más de una letra
Este tipo de variable se almacena con los bits 5 y 7 de la primera letra en el nombre fijado a 1 y el bit 6 de la primera letra en el nombre se reinicializa a 0. Los caracteres subsiguientes en el nombre se almacenan como el código del carácter, con todo pero el último carácter en el nombre teniendo el bit 7 se reinicializa a 0. El final del nombre está indicado por el último carácter en el nombre teniendo el bit 7 fijado a 1. En decimal esto significa que la primera letra en el nombre se almacena como el código de la letra +64, los caracteres subsiguientes son almacenados como el código de la letra/número, con el último carácter siendo el código del carácter +128. Incidentalmente, puede llegar a confundirse mezclando el decimal y el binario en su mente, así que si Vd. puede, es mucho mejor en binario ya que estamos tratando bits individuales.

Aquí viene un diagrama para ilustrar esto:





Matriz de números
Una matriz de números se reconoce por los bits 5 y 6 siendo 0 y el bit 7 siendo 1 en el byte que contiene el nombre de la matriz numérica. Así pues, el nombre está contenido como el código del nombre menos 96 en los bits del 0 al 4. Esto corresponde a los ocho bits del código del nombre +32.

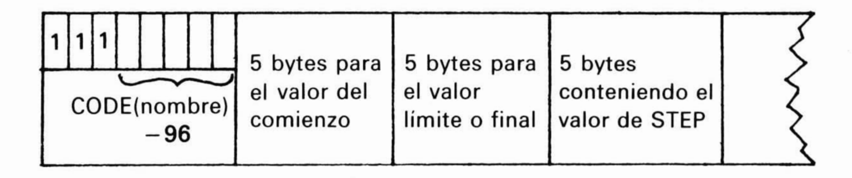
1 0 0 — CODE(nombre) −96	2 bytes para la longitud total de las dimensiones de los elementos y 1 para el byte que contiene el número de dimensiones	1 byte conteniendo número de dimensiones especificadas	

2 bytes conteniendo el número de los elementos en la primera dimensión	2 bytes conteniendo el número de los elementos en la última dimensión	5 bytes para cada elemento de la matriz	

Los elementos de la matriz están contenidos en una forma lógica, de forma que los elementos del subíndice de la primera matriz viene primero, después todos los del subíndice de la segunda dimensión y así sucesivamente.

La variable de control de un bucle FOR...NEXT

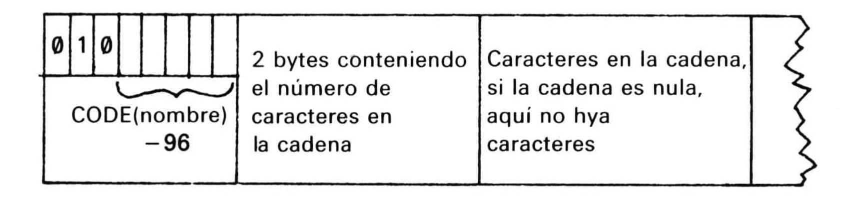
Las variables de control del bucle FOR... NEXT se reconocen porque tienen los bits 5, 6 y 7 en su nombre colocados todos a 1. La letra otra vez se guarda en los bits del 0 al 4 como el código (nombre) – 96. En decimal, sobre los ocho bits o el byte como un todo, esto sería el código (del nombre) + 128. Hay un byte para el nombre, 5 bytes para el valor del comienzo (el anterior al TO en la línea del programa fijando el bucle), cinco bytes para el valor límite (el número después del TO) y cinco bytes para el valor de STEP. Allí siguen dos bytes que dicen a qué número de línea el bucle hace un "bucle" y otro byte para el número de la sentencia con esa línea a la que el bucle salta. Esto hace un total de 19 bytes ocupados en el área de la memoria.



2 bytes conteniendo el número de línea a la que el bucle salta	1 byte conteniendo el número de sentencia en la línea a la que el bucle salta

Variables de cadena

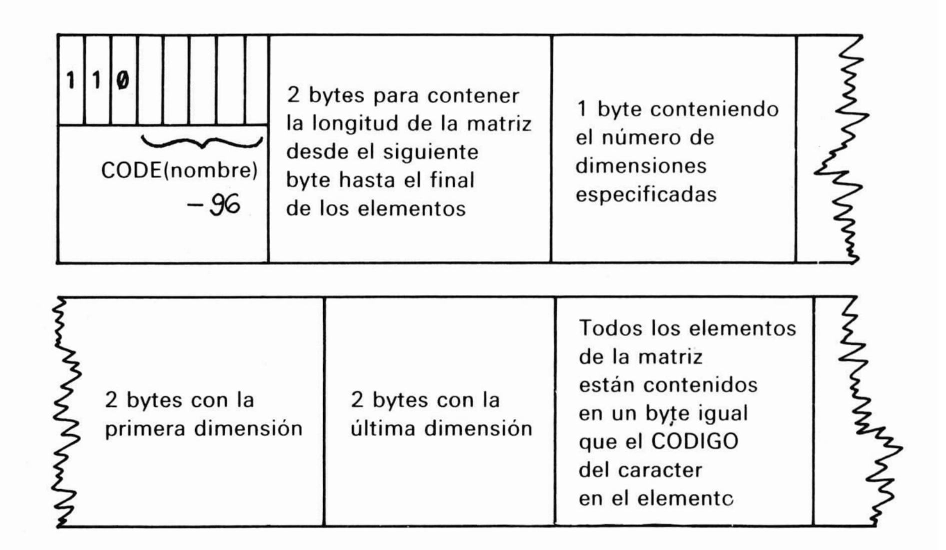
Las variables de cadena se reconocen porque tienen los bits 5 y 7 que se reinicializan a 0 y el bit 6 colocado a 1. El nombre (sólo una letra) se almacena en los bits 0 al 4 como el código del nombre menos 96. En decimal esto significaría que el byte que contiene el nombre de la variable de cadena tendría un valor del código (nombre) –32. Hay un byte para el nombre seguido por dos bytes para la longitud de la cadena, o sea, cuantos caracteres contiene la cadena, como la función LEN del BASIC. Entonces aparecen caracteres en la cadena, almacenados simplemente como sus códigos. Si la cadena es nula, aquí no hay caracteres.



Matrices de caracteres

Las matrices de caracteres se reconocen porque tienen el bit 5 que se reinicializa a 0 y los bits 6 y 7 que son colocados a 1. El nombre se almacena en un byte que consiste de una sola letra. Los bits del 0 al 4 contienen el CODE del nombre menos 96. En decimal, esto corresponde sobre todo el byte al código del nombre + 96. Hay un byte para el nombre, después dos bytes que contienen la longitud del resto de la cadena incluyendo los elementos, las dimensiones y uno extra para cuantas direcciones hay allí. Los bytes de longitud van seguidos de un byte que contiene el número de dimensiones determinadas. Si Vd. tenía DIM A$(4,5,6) este byte contendría el 3. Las propias dimensiones siguen en dos bytes cada una por el orden en que fueron escritas en la línea del programa determinando el DIM. Así pues, en el ejemplo del DIM A$(4,5,6) Vd. obtenía, después del 193 para la matriz del nombre, los números 127,0,3,4,0,5,0,6,0

seguidos por los códigos de los caracteres en los elementos de las matrices. Naturalmente, los ceros son debidos al uso de dos bytes en algunos casos como descritos. Finalmente, viénen los elementos de la matriz de caracteres. Estos se almacenan por sus códigos en un byte cada uno y por orden.



Edición del comando/línea, datos INPUT y el espacio de trabajo
Usado por el computador durante la ejecución de los programas en BASIC, edición e información del teclado. Estas áreas pueden extenderse o retraerse a nada dependiendo en cuanto espacio actualmente necesita. Generalmente, encontrará que si Vd. examina estas áreas no hay nada allí ya que todo ha sido verificado de forma que no se usa ningún espacio si no se necesita.

Stack del calculador
El stack del calculador se usa para colocar los números de punto flotante, los enteros y la información sobre las cadenas. generalmente todo va como cinco bytes. Los números en su formato de cinco bytes y las cadenas como cinco bytes de detalles sobre ellos (comienzo de las direcciones, etc.). Las operaciones aritméticas y otros hacen uso de este stack.

Memoria de reserva
Es la parte de la memoria no utilizada entre el stack del calculador y el de la máquina. Puede llegar a utilizarse si los stacks crecen o se hacen cambios para las variables o programa, etc. No es muy seguro guardar allí cosas de importancia.

Stack de la máquina y stack de GOSUBs
El stack de la máquina es donde el Z80A puede poner la información como valores de registros. El stack del GOSUB conserva los datos suficientes para que el RETURN regrese al punto apropiado en un programa en BASIC después de una subrutina en BASIC. La información sobre el número de la línea y sentencias que hay a continuación del GOSUB, se almacena aquí para saber adónde debe dirigirse.

RAMTOP
Esto es el límite de la memoria usada por el BASIC, aparte de los datos de gráficos definidos por el usuario que se guarda entre el RAMTOP y el final de memoria disponible. Todo lo que se salve sobre el RAMTOP en la RAM está seguro de todo lo que esté en BASIC excepto el POKE. El NEW sólo funciona mientras duren las direcciones contenidas en la RAMTOP, de forma que las especificaciones de los UDGs son seguras a menos que desactive la máquina o un POKE. Lo mismo se aplica a cualquier dato o código de máquina de la RAMTOP, si todavía la salvó desde el NEW.

OTRAS VERSIONES DEL BASIC

Si Vd. desea convertir un programa escrito en otra versión del BASIC, esta sección puede que le resulte interesante. Le muestra cómo puede emular ciertos comandos, funciones y otras características encontradas en ciertas versiones del BASIC que no pueden ser utilizadas directamente en el BASIC del Spectrum Sinclair. Algunas veces, se trata de usar otra palabra o algunas veces puede ser necesaria una rutina completa.

Matrices
Con las matrices numéricas, el único problema que puede encontrarse son los subíndices. Estos pueden comenzar en 0 dónde los subíndices del Spectrum comienzan con el 1. Generalmente la solución es añadir un 1 a todas las referencias para cualquier subíndice de matriz, incluyendo las sentencias DIM, si el programa hace uso del subíndice 0. Observe atentamente ciertos detalles como A(G*3–1) que con un truquillo pueden convertirse. Tendrá que convertirlos individualmente conforme se los vaya encontrando.

ASC
Esto corresponde a la función CODE del Spectrum, por lo menos para los caracteres con los códigos 32 al 126 en el Spectrum. Si se usa para almacenar datos en una cadena o matriz y a continuación se ha de codificar, sería mejor que no hiciera ningún cambio; por el contrario estudie la rutina para encontrar los valores o caracteres que difieren de su equivalente en el Spectrum. Cualquier cambio debe realizarse individualmente para cada programa ya que no existe ninguna regla para los caracteres que tienen los valores CODE/ASC fuera del rango entre 32 y 126. Puede que existan tan sólo una o dos diferencias con este rango.

CALL
Se usa para saltar dentro del código de máquina y puede reemplazarse por el USR. El Spectrum retorna un número cuando viene del BASIC, el cual deberá encontrar algo con el que hacer, o sea, la dirección CALL podría reemplazarse por la dirección LET A = USR o la dirección RANDOMIZE USR.

Grados y Radianes
Las funciones trigonométricas del Spectrum trabajan con radianes. Si Vd. necesita adaptar una rutina usando grados pueden cambiarse a radianes con:

LET RADIANES=(PI*GRADOS)/180

DIM
El programa puede llamar a varias matrices a colocar usando una sentencia DIM separada por comas. Así pues, reemplazaría DIM(A3),B(4),C(5) con DIM A(3): DIM B(4): DIM C(5). Asegúrese de que las sentencias DIM determinan matrices de cadenas o de caracteres, porque puede que Vd. necesite un subíndice extra en el Spectrum, ya que estas matrices están todas determinadas a una longitud fija en la sentencia DIM. Generalmente en otros computadores, la longitud de cada cadena en la matriz es solamente tan larga como necesite serlo. Así pues, si el programa llama al DIM A$(4) esto significa cuatro cadenas separadas, no una cadena de cuatro caracteres como esta sentencia puede aparecer a algunos usuarios del ZX. La forma de convertir esto para el Spectrum es pensar qué cadena es la más larga, digamos 10, letras y acoplarla al DIM del Spectrum de acuerdo al DIM A$(4,10).

DIV
Esta función retorna parte del número después de una división, de forma que 10 DIV 4 sería 2. Reescriba esto de esta forma INT(10/4), es decir, ponga la división entre paréntesis y añada el INT.

DO...UNTIL
El programa realiza la ejecución de parte del programa entre el DO y UNTIL y sólo se detiene una vez se ha realizado la condición después del UNTIL. Generalmente, Vd. convertiría esto con IF...THEN GOTO...

DRAW
La mayoría de los BASICs dibujarían una línea especificando los finales de las coordenadas, o sea, DRAW 200,90 dibujaría una línea terminando en los ejes 200 horizontal y el 90 vertical. Esto causaría problemas porque la rutina DRAW del Spectrum esperaría valores x e y relativos, o sea, el eje x a través de la pantalla desde donde comienza la línea y el eje y desde donde comienza la línea. Observe que la mayoría de los BASICs moverían el cursor PLOT para después dibujar a partir de allí. Así pues, para convertir MOVE a,b:DRAW x,y pero para el Spectrum Vd. usaría PLOT a,b:DRAW (x–a),(y–b). Observe que hay una diferencia en que el PLOT tiene que utilizarse dos veces, una para el MOVE y la otra para sombrear en el primer punto de la línea.

ELSE
Es una parte de la instrucción IF...THEN...ELSE en la cual la acción que está después de ELSE se ejecuta si la condición después de IF es falsa. Ejemplo: IF X = 1 THEN PRINT "X = 1" ELSE PRINT "X no es 1". En el Spectrum se escribiría:

IF X = 1 THEN PRINT"X = 1"

IF X < > 1 THEN PRINT"X no es 1"

También puede reescribir algunos ejemplos usando el AND para unir las condiciones. Sin embargo, usando las dos sentencias IF...THEN es la mejor forma de seguirle todo normalmente.

END
Bajo algunas condiciones éste puede ser omitido. En todos los aspectos es lo mismo que el STOP.

Exponente
BASICs diferentes usan símbolos diferentes para esto; tan sólo reemplace los ** o ^ por la ↑ del Spectrum.

Bucles FOR... NEXT
La diferencia que puede encontrar es que el programa verifica si el bucle ha sido terminado. Si la verificación se hace al final del bucle, en el NEXT, esto significa que el bucle será ejecutado por lo menos una vez. En el Spectrum, la verificación se hace después del FOR, significando que el bucle nunca se ejecutará si el valor final sobrepasa el valor del comienzo siendo mayor que el valor antiguo.

GET y GET$
Esto lee el teclado y generalmente espera a que se pulse una tecla antes de que el programa se mueva. La expresión completa es parecida a LET A = GET o GET A (y lo mismo para GET$). Esto puede convertirse usando el INKEY$ (si el computador espera a que se pulse una tecla, Vd. ha avanzado de manera especial en sus conocimientos) o usando INPUT (debía ser obvio que necesitaba el ENTER).

Si Vd. usa el INKEY$ para convertir LET A = GET o GET A con:

1000 LET A = CODE INKEY$: IF A = 0 THEN GOTO 1000

Con el GET$ (o sea, LET A$ = GET$):

1000 LET A$ = INKEY$: IF A$ = ""THEN GOTO 1000

IF...THEN...
Algunos BASICs permiten que el THEN pueda ser omitido. Sin embargo, debe ser incluido en el Spectrum. Por ejemplo, IF X = 1 PRINT "Uno" debe escribirse como IF X = 1 THEN PRINT "Uno"

INKEY
Esto lee el código de la tecla que se pulsó recientemente en el teclado. Puede ser convertido por CODE INKEY$ de forma que LET A = INKEY se convirtiera en LET A = CODE INKEY.

Puede encontrarse una versión con un número entre paréntesis. Este es el momento que la función esperará a que se pulse una tecla antes de moverse. La forma más fácil es poner un PAUSE antes de buscar el teclado. Así pues:

LET A = INKEY(50)
puede convertirse:

PAUSE 50:LET A = CODE INKEY$

Si las unidades de tiempo no son las mismas, deben necesariamente hacer algunos cambios sobre la longitud del PAUSE.

INSTR
Esta función busca una copia de una cadena con otra. Necesitará una rutina completa para reemplazar esto. La rutina que se muestra produce resultados similares al LET Y = INSTR(B$,C$). El B$ es la "gran" cadena en la cual Vd. busca una copia de la string C$ más pequeña. No sucede ningún error si tanto una cadena como otra es mayor o quizás del mismo tamaño.

```
9200 REM INSTRUCCIONES
9205 LET P=0
9210 IF LEN C$=0 OR LEN B$=0 OR
LEN C$>LEN B$ THEN RETURN
9215 FOR P=1 TO LEN B$-LEN C$+1
9220 IF B$(P TO P+LEN C$-1)=C$ T
HEN RETURN
9225 NEXT P
9230 LET P=0
9235 RETURN
```

Volviendo a la rutina, Y contendrá la posición donde comienza la copia, 1 si la copia comienza al principio, 2 si comienza en la segunda letra y así sucesivamente. Si no se encuentra ninguna copia o el C$ es mayor que el B$, entonces Y será 0. Por la razón que sea, si Y es 0, signfica que no hay una copia exacta de C$ en B$.

Variables enteras
Generalmente son identificadas porque tienen el símbolo del tanto por ciento % al final del nombre. Por ejemplo, A%. Normalmente, Vd. podría usar cualquier nombre de variable que quiera asegurándose de que el valor que va en la variable es un entero, o sea, si el original LET A% = 3/2 y Vd. quiere imprimir PRINT A%, obtendrá un 1. Para tener el mismo resultado en el Spectrum Vd. debería usar LET A=INT(3/2).

LETF$
La función LEFT$(A$,B), la cual toma los primeros B caracteres desde el A$ puede escribirse como A$(TO B) en el Spectrum. Así pues, algo así como LET R$=LEFT$(A$,B) se convertiría en LET R$ = A$(TO B).

LINK
Esto es una llamada a una rutina en código de máquina. Puede reemplazarlo por USR.

Expresiones lógicas
Parece que va a encontrarse con más de un problema con los valores de TRUE y FALSE. El cero siempre es valor falso en los programas que se encontrará, pero verdad puede a menudo ser –1. Con esto quiero decir que algo como PRINT (1=2) imprimirá 0 (igual como lo hará en el Spectrum) y PRINT(1 = 1) imprimirá –1 porque la expresión es verdadera. En el Spectrum PRINT (1 = 1) imprimirá 1. Aquí la solución es negar el resultado de estas expresiones. Si el original lee:

LET X = 10 – (PUNTOS = 6) + (TIEMPO < 300)

esto sería reescrito para el Spectrum así:

LET X= 10 + (PUNTOS = 6) – (TIEMPO < 300)

Sea consciente del uso del AND y el OR como operadores binarios los cuales operan sobre números bit por bit. Generalmente esto se manifiesta como en esta corta sentencia:

LET A = A AND 4

No hay ninguna forma más fácil para convertir estos en BASIC. Deberá reescribir la rutina generalmente o olvidarse de ella por completo.

LOGS
El Spectrum usa los logaritmos naturales. Si Vd. necesita logs para otras bases (normalmente para la base 10) use:

LET LOGBASEX número = LN número / LN X

donde X es la base y número es el número que Vd. desea para encontrar el logaritmo.

MAT
Abreviación para Matriz o Matrices. Esta es una función la cual trabaja con todos los elementos de una matriz:

```
10 DIMX(10)
20 DIMY(10)
30 MATX=Y
```

O sea, todos los elementos de la matriz X son hechos igual a los de la matriz Y. Vd. deberá usar un bucle para realizar la operación con todos los elementos de las matrices:

```
10 DIM X(10)
20 DIM Y(10)
25 FOR M = 1 TO 10
30 LET X(M) = Y(M)
35 NEXT M
```

MID$
MID$(W$,T,U) toma U caracteres de la cadena W$, comenzando con el elemento T. En el Spectrum use W$(T TO T + U - 1).

MOD
A MOD B nos da el resto después de que A ha sido dividida por B. Reemplace con A – (INT(A/B)*B).

MOVE
Todo lo que esto hace es alterar la posición del cursor PLOT sin dibujar nada en la pantalla. Generalmente usado para determinar un punto desde el cual se dibuja una línea, y para examinar una parte particular de la pantalla. Observe que cuando se usa para dibujar líneas, la función asociada DRAW llenará el primer punto de la línea El DRAW del Spectrum no lo hará y Vd. deberá usar el PLOT y DRAW juntos. Para simular MOVE tiene que hacer un POKE de los valores de la X y de la Y dentro de las variables del sistema que contienen las

actuales coordenadas de PLOT. Para reemplazar el MOVE a la misma escala (o sea MOVE X,Y) use:

POKE 23677,X:POKE23678,Y

NEXT
En algunas versiones del BASIC puede omitirse el nombre de la variable después del NEXT. Si es así, se incrementa el valor de la variable de control más reciente. Esto no es posible en el Spectrum. Vd. siempre debe especificar el nombre de las variables de control.

ON...GOTO/GOSUB...
Esto normalmente toma la forma:

ON X GOTO 200,300,400,500

Que significa que si la X fuera uno, el programa haría un GOTO al primer número de línea en la lista después del GOTO o GOSUB. Si X fuera 2, el programa haría un GOTO/GOSUB del segundo número de línea y así sucesivamente.

La forma más fácil es convertir usando IF...THEN GOTO... de forma que lo de arriba se convertiría en:

IF X = 1 THEN GOTO 200
IF X = 2 THEN GOTO 300
IF X = 3 THEN GOTO 400
IF X = 4 THEN GOTO 500

Vd. podría usar AND para convertirlo:

GOTO (200 AND X = 1) + (300 AND X = 2) + (400 AND X = 3) + (500 AND X = 4)

Si los números de línea van a un paso que se lo permita, use un GOTO computado como éste:

GOTO 100 + (X * 100)

PAINT
Esto es un comando de gráficos que llena una área de la pantalla en color. No existe ninguna forma rápida para convertir esto en BASIC y de todas formas varía de una máquina a otra. Esto y muchos otros comandos de gráficos es mejor evitarlos porque los gráficos son probablemente el área donde hay las más grandes diferencias entre varios computadores.

PEEK y POKE
Esta es la otra gran diferencia de un computador a otro. Generalmente se necesita el conocimiento de la arquitectura del computador y que a menudo es imposible convertir.

PRINT
En algunos computadores se usa el signo de interrogación ? como abreviación del comando PRINT, el cual debe escribirse completo en el Spectrum.

PROC,ENDPROC
PROC es la abreviatura de procedimiento, lo cual es una forma de subrutina llamada más por el nombre que por el número de línea.
En el Spectrum Vd. reemplazaría el PROC con una subrutina, llamada GOSUB (número de línea). Reemplace el ENDPROC con un RETURN.

NUMEROS ALEATORIOS
Algunas computadoras generan números aleatorios con la expresión RND (X) la cual retorna un número completo entre el 1 y la X inclusive. Convierta el RND(X) con INT(RND*X)+1. Si Vd. ve una expresión como RND (0) generalmente significa repetir el último número aleatorio. El RND(–X) generalmente significa lo mismo que RAND en el Spectrum.

REPEAT...UNTIL
Esto significa la creación de un bucle sin referencia de los números de línea. El bucle sigue funcionando para siempre, a menos que la condición mencionada después de la palabra UNTIL sea verdadera y el programa termina el bucle. A continuación tiene un ejemplo de su uso:

```
10 LET X=0
20 REPEAT
30 LET X=X+1
40 UNTIL X=10
```

Esto puede ser convertido usando una sentencia IF...THEN GOTO... en la línea que contiene el UNTIL, el GOTO refiriéndose a la sentencia o línea que viene después de la que contiene el REPEAT. Si Vd. lo convierte como yo lo he hecho aquí, puede también incluir una sentencia REM como un recordador y haga esto:

```
10 LET X=0
20 REM AQUI EL BUCLE
30 LET X=X+1
40 IF X<10 THEN GO TO 20
```

RESET
Se usa para hacer un punto o bloque en la pantalla blanca o negra. Es un comando de gráficos que normalmente puede reemplazarse con el PLOT o PRINT dependiendo del computador y de la naturaleza del programa.

RESTORE
El Spectrum puede tener un número de línea después del comando RESTORE y esto puede muy a menudo simplificar un programa. Es bastante común encontrarse con algo así:

```
100 RESTORE:FOR A = 1 TO 4: READ A$: NEXT A
110 DATA "PESCADO","PAJAROS","MAMIFEROS","REPTILES"
120 DATA "GATOS","PERROS"
```

Es necesario saltarse ciertos datos hasta que llega a la sección correcta. En el Spectrum, reemplace todo en la línea 100 con:

```
100 RESTORE 120
```

RIGHT$
RIGHT$(R$,X) toma los caracteres X de la derecha de la cadena R$. En el Spectrum use R$(LEN R$ – X + 1 TO). Observe que no es necesario poner nada después del TO en la versión del Spectrum.

SCROLL
El comando de deslizamiento del ZX81 puede reemplazarse con el PRINT AT 21,31" (observe los dos apóstrofes). Esto tiene la ventaja de ser en un BASIC muy simple pero tiene la desventaja de inutilizar el mensaje "scroll?" del Spectrum. La llamada de la ROM LET A = USR 3582 deslizará la pantalla sin que este efecto suceda. Naturalmente tiene la desventaja de que al usar una ROM diferente las direcciones pueden ser diferentes.

SET

Ver RESET.

TAB(X,Y)
Aparte de que el programa puede haber sido diseñado para un computador con más caracteres a lo ancho de la pantalla (o menos), esto es lo mismo que AT Y,X; en el Spectrum.

THEN
El THEN puede ser omitido en algunos computadores, es decir IF X = 2 GOTO 10, pero debe ser incluido en el Spectrum, así IF X = 2 THEN GOTO 10.

Variables indefinidas
Si un programa intenta utilizar una variable antes de que ésta haya sido determinada,la mayoría de los BASICs asumirán un valor de cero. Sin embargo, en el Spectrum se genera un error "Variable not found".

VAL
Igualmente, si la cadena a la que se le aplica el VAL no es numérica el valor que se obtiene es 0. En el Spectrum, Vd. puede obtener diferentes errores (o sea, el error 2 de arriba o el error C sin ningún sentido en el BASIC).

COMO FUNCIONA LA VISUALIZACION EN PANTALLA

Después de leer el capítulo 24 del manual del Sinclair puede ser ya consciente de lo inusual del diseño del patrón de puntos de la visualización en la pantalla. La copia del cuadro en la memoria se almacena en dos bloques. Primero el patrón de puntos almacenado en un bloque de 6144 bytes en la dirección 16384 a la 22527. Cada punto en la pantalla tiene un bit correspondiente (ocho bits en cada byte, 6144*8 = 49152 puntos/bits) pero los puntos no están en el mismo orden en la pantalla que en la memoria. Esto será explicado más adelante.

El segundo bloque contiene la información del color, brillo y destello en un bloque de 768 bytes en la dirección 22528 a la 23295. Echaremos una mirada a ambas secciones. Primero, el fichero para la visualización que contiene el patrón de puntos de la pantalla. Recuerde que cada bit corresponde a un punto en la pantalla y cuando hacemos un POKE de todo un byte estamos cambiando los ocho bits/puntos al mismo tiempo. Cualquier bit que se determine a 1 será el color INK en la pantalla y cualquier bit que esté determinado a 0 aparecerá con el color de PAPER. Si el valor con el que nosotros hacemos un POKE es 255 (que es BIN 1111 1111 que fija los ocho bits a 1), cada uno aparecerá como una línea con el ancho de un carácter y un punto de altura.

```
10 FOR A=16384 TO 22527
20 POKE A,255
30 NEXT A
```

Habrá ya observado varias cosas. Primero de todo, no parece ser un patrón reconocible en la forma como llena la pantalla. Segundo, la pantalla se va llenando en tres bloques, las ocho hileras de arriba (imprime las hileras de la 0 a la 7), las ocho hileras de la mitad (imprime las hileras de la 8 a la 15) y finalmente las ocho hileras de la parte inferior (imprime las hileras 16 a la 23).

Esto significa que la pantalla inferior también se almacena o guarda allí. Lo que sucede es que se llena la hilera 0 de la primera línea de puntos de la parte superior, a continuación la línea de puntos de la hilera 1, y así sucesivamente hasta llegar a la hilera 7. Entonces se llena la hilera 0 de la segunda línea de puntos, después la hilera 1 de la segunda línea de puntos, y así hasta completar las hileras de la 0 a la 7 de las ocho líneas de puntos. De todo esto podemos lle-

gar a la conclusión que para encontrar la dirección de la línea superior de puntos en una hilera de caracteres (usando PRINT AT Y,X; coordenadas) en la hilera Y, columna X. Podríamos usar:

```
LET DIRECCION=16384+Y*32+X
```

Esto sólo funcionaría para los valores de Y desde 0 al 7. Habrá observado que el bloque de en medio de las hileras 8 se llenan de la misma forma que la hilera de la parte superior una vez ha sido terminada. Desafortunadamente, la ecuación sólo funciona para las hileras 0 a la 7 de la parte superior. Sin embargo, esta ecuación le dirá dónde hacer un POKE para la línea superior de puntos en las hileras 8 a la 15:

```
 LET DIRECCION=16384+2048+(y-16)
*32+x
```

El número 2048 indica cuántos bytes representan las hileras desde la 0 a la 7. Recuerde, hay 32 columnas a través de la pantalla, 8 bytes para la posición de cada carácter y 8 hileras en ese bloque de la pantalla (32*8*8 = 2048). Igualmente, necesitamos otra ecuación para el tercer bloque más inferior de la pantalla:

```
 LET DIRECCION=16384+4096+(y-16)
*32+x
```

Este podría utilizarse como esto. Esto es un programa ejemplo que imprime el número 8 debajo, a la izquierda de la pantalla y después hace un POKE de una línea arriba de cada figura 8 como si subrayara sólo la parte superior del carácter:

```
  10 LET X=0
  20 FOR Y=0 TO 21
  30 PRINT AT Y,0;"8"
  40 IF Y<=7 THEN LET DIRECCION=
16384+Y*32+X
  50 IF Y>=8 AND Y<=15 THEN LET
DIRECCION=16384+2048+(Y-8)*32+X
  60 IF Y>=16 THEN LET DIRECCION
=16384+4096+(Y-16)*32+X
  70 POKE DIRECCION,255
  80 NEXT Y
```

Es un incoveniente del bit tener que escribir tres líneas de programa (de la 40 a la 60) cuando con una se podría hacer. La parte más matemática que haya podido ver la encontrará resumida en una sola línea:

```
  LET  DIRECCION=16384+2048*INT
 Y/8)+32*Y-(INT (Y/8)*8*32)+X
```

Después de una especie de juegos de manos de bits aparece:

```
LET DIRECCION=16384+2048*INT (
Y/8)+(Y-(INT (Y/8)*8))*32+X
```

O:

```
LET DIRECCION=16384+2048*INT (
Y/8)+32*Y-256*INT (Y/8)+X
```

O:

```
LET DIRECCION=16384+1792*INT (
Y/8)+32*Y+X
```

Y finalmente:

```
LET DIRECCION=16384+32*(56*INT
(Y/8)+Y)+X
```

Parece extraño, pero funciona. Así pues, si le cambiamos las líneas 40, 50 y 60 a nuestra maravillosa línea:

```
  10 LET X=0
  20 FOR Y=0 TO 21
  30 PRINT AT Y,0;"8"
  40 LET DIRECCION=16384+32*(56*
INT (Y/8)+Y)+X
  50 POKE DIRECCION,255
  60 NEXT Y
```

Funciona muy bien. Ya puede ir pensando qué hacer con las otras líneas que no sea la de la parte superior en cualquier carácter. Lo que se produce es que como la línea superior de cada bloque de ocho hileras es almacenada una detrás de otra, entonces la segunda línea de puntos en cada hilera siguen 256 bytes (32 caracteres por hilera *8 hileras = 256) más o menos.
Aquí tenemos la fórmula para obtener la dirección de un punto de la pantalla que se encuentre en cualquiera de los tres bloques de 8 hileras.

```
LET DIRECCION=16384+32*(56*INT (
Y/8)+Y)+X+ LINE *256
```

Pruebe este programa de demostración que llena una línea entera una columna de ochos:

```
  10 LET X=1
  20 FOR Y=0 TO 21
  30 PRINT AT Y,0;"8"
  40 FOR L=0 TO 7
  50 LET DIRECCION=16384+32*(56*
INT (Y/8)+Y)+X+L*256
  60 POKE DIRECCION,255
  70 NEXT L
  80 NEXT Y
```

Pruebe este programa, el cual, muy despacio e ineficientemente, llena la pantalla haciendo POKEs de las direcciones en el fichero de visualización en el orden correcto para llenar cada posición de los caracteres en el orden en que la persona espera verlo desde la pantalla. Incidentalmente, extendiendo el valor de X a 23 significa que Vd. puede hacer un PLOT en las líneas 22 y 23 en donde normalmente no lo podría hacer. Sin embargo, observe que Vd. no puede hacer POKEs de puntos individuales sin afectar a los ocho puntos controlados por todo el byte, a menos de que esté preparado para revisar los valores que ya están en el byte y determinar relacionadamente qué valor ha de POKEar.

```
   10 FOR Y=0 TO 23
   20 FOR X=0 TO 31
   30 FOR L=0 TO 7
   40 LET DIRECCION=16384+32*(56*
INT (Y/8) +Y) +X+L*256
   50 POKE DIRECCION,255
   60 NEXT L
   70 NEXT X
   80 NEXT Y
```

En binario, hay un método más fácil de entender para mostrar el fichero. Ciertos grupos de bits proporcionan cierta información relativa a la posición para mostrar el fichero. Esto lo encontrará de utilidad en código de máquina ya que le permite el acceso de texto y gráficos a la pantalla bastante fácilmente. Este diagrama le mostrará cómo esto está organizado.

bit 15	bit 14	bit 13	bit 12	bit 11	bit 1Ø	bit 9	bit 8
Ø	1	Ø	Estos dos bit son Ø si el punto referido está en el tercio de la pantalla, 1 si está en la mitad del tercio ó 2 si está en el tercio superior		el caracter de 8 x 8. Valor del bloque Ø para la columna superior, 7 para la inferior		
El valor fijo para apuntar el inicio del fichero de visualización BIN es 16384							
16384			2Ø48*INT(Y/8)		256 * LINE		

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit Ø
En el tercio apropiado de la pantalla, que filas de caracteres de la Ø a la 7 –de forma que para Y = 8 ó Y = 16 esto sería Ø Y–INT (Y/8) 8			Coordinada X del bloque de caracteres a través de la pantalla, Ø a 31				

Junte las expresiones de cada sección en el diagrama y obtendrá una experiencia que ya le será familiar:

16384 + 2048 * INT(Y/8) + Y – (INT(Y/8) * 8) + X + LINE*256

Segundo, el fichero de atributos. Esto es más fácil de entender. La información del color/flashing/brillo para cualquier posición de caracteres se almacena toda en un byte. Hay 32 por 24 casillas de caracteres en la pantalla haciendo un total de 768 bytes. Estos son almacenados en un bloque de la memoria llamado el fichero de atributos desde la dirección 22528 a la 23295. El diseño de estos bytes es bastante simple para entender comparado con la visualización del fichero.

El primer byte (22528) corresponde al carácter de la parte superior izquierda de la pantalla de TV. Los primeros 32 bytes corresponden a la hilera superior de 32 caracteres a través de la pantalla. Los siguientes 32 bytes corresponden a la segunda hilera de 32 caracteres a través de la pantalla de TV, los siguientes 32 bytes corresponden a la tercera hilera de 32 caracteres y así sucesivamente hasta llegar al último carácter de la pantalla en la hilera inferior (fila 23). Si quiere ver lo fácil que es de entender, pruebe este programa que hará que todo lo de la pantalla aparezca en negro. Observe el orden por el cual la pantalla se llena, incluyendo las dos líneas del final. Aunque éstas aparecen negras por un momento, el indicador del código del reporte sobreescribe esto. Añada un PAUSE 0 al final de la línea 30 para ver el efecto y pulse cualquier tecla (excepto las SHIFTS!) para terminar el programa:

```
10 FOR A=22528 TO 23295
20 POKE A,0
30 NEXT A
```

La información en un byte de atributos está contenida en un formato que permite a los bits individuales almacenar valores para el flashing, brillo del color PAPER y color INK:

bit 7	bit 6	bit 5,4 & 3	bit 2,1 & Ø
FLASH 1 si destella Ø si no destella	BRIGHT 1 ni tiene brillo Ø normal	Color PAPER en binario	Color INK en binario

Con esto, es posible hacer un POKE en cualquier posición del fichero de atributos si Vd. conoce el PRINT AT Y,X; coordenadas (donde Y es la posición PRINT en vertical con respecto a la pantalla y la X es lo mismo pero en horizontal).

```
POKE (22528+32*Y+X),NUMERO
```

Esto también podría hacerse usando PRINT AT Y,X;OVER1;""; pero es otra historia.

Pruebe este programa. Le pedirá que entre los valores de Y y X, imprimir un carácter allí, y después le pedirá que entre un valor para los atributos y Vd. verá si cambia de color, brillo y/o flashing depende de lo que Vd. haya entrado. Del diagrama de arriba, quizás quiera usar el BIN para entrar el valor de atributos, es decir, si Vd. quisiera no flashing, brillo, PAPER blanco y caracteres INK azules, Vd. entraría BIN 01111001:

```
  10 INPUT "ENTRA EL VALOR PARA
X";X
  20 INPUT "ENTRA EL VALOR PARA
Y";Y
  30 PRINT AT Y,X;"+"
  40 INPUT "ENTRA EL VALOR PARA
ATRIBUTOS";ATRIBUTOS
  50 POKE (22528+32*Y+X),ATRIBUT
OS
```

No existe ningún problema en hacer un PEEK del fichero de atributos ya que ATTR puede hacerlo por Vd., pero si ya está determinado use esto:

```
LET P=PEEK (22528+32*Y+X)
```

LLAMADAS DEF FN UTILES

Aquí verá algunas llamadas de función que yo he encontrado útiles y además todas juntas para Vd.

(1) Para encontrar si el bit B (0 al 7) del número N (0 al 255) es 1 o 0:

DEF FN A(B,N) = INT ((N-INT(2 ↑ (B+1)))*(2 ↑ (B+1)))/(2 ↑ 3)

(2) Añada un porcentaje P a una cantidad A.

DEF FN S(A,P) = A * P/100 + A

(3) Dada una cantidad total T que incluye un porcentaje P añadido a una cantidad original, esta función le dirá cuál era la cantidad original. Util para los cálculos del VAT, etc.:

DEF FN A(P,T) = (T*100)/(100 + P)

(4) Impar o par. FNO() será 1 si el número N es impar, o cero si es par:

DEF FN O(N) = N – INT (N/2) * 2

(5) Para encontrar el color PAPER en la posición de la pantalla Y,X (es decir, resolver los atributos a componentes):

DEF FN P(Y,X) = INT ((ATTR (Y,X) – INT (ATTR (Y,X)/64)*64)/8)

(6) Para encontrar el color INK en la posición Y,X de la pantalla:

DEF FN I(Y,X) = INT(ATTR(Y,X) – INT (ATTR(Y,X)/8)*8)

(7) Para encontrar el estado FLASH (0 o 1) en la posición Y,X de la pantalla:

DEF FN F(Y,X) = INT (ATTR(Y,X)/128)

(8) Para encontrar el estado de BRIGHT en la posición Y,X de la pantalla:

DEF FN B(Y,X) = INT ((ATTR(Y,X) – INT(ATTR(Y,X)/128) *128)/64)

(9) Un PEEK de 2 bytes comenzando en la dirección A:

DEF FN P(A) = PEEKA + 256 * PEEK (A + 1)

Vd. podría usar esto en su programa mientras lo escribe. En cualquier momento, PRINT FN(23641) – 16384 le dará una vaga idea de cuánta memoria ha sido utilizada (incluyendo todo hasta el final de las variables). Esto le ahorra escribir una expresión muy larga del PEEK todo el tiempo.

(10) Lo que queda de memoria en bytes:

DEF FN M() = PEEK 23730 + 256 * PEEK 23731 – PEEK 23653 – 256 * PEEK 23654

(11) Redondeo del valor de X al número más cercano:
DEF FN W(X) = INT(X + 0.5). Esto sufre desde el punto de vista que el Spectrum contiene el número 0.5 de tal forma, que dando a X un valor de 0.5 dará 0. Esto puede corregirse incrementando el valor añadido a la X en la función, digamos, 0.50000001.

(12) Números aleatorios entre 1 y X:

DEF FN R(X) = INT (RND * X) + 1

(13) Centrando una cadena en medio de la pantalla. Esto retorna el valor para TAB para imprimir la cadena de forma que está en el medio de la pantalla:

DEF FN M(M$) = (16 – LEN M$ / 2) AND LEN M$ < 33

Esto sólo funciona para cadenas de hasta 32 caracteres de longitud. Si son más largas de 32 caracteres, la rutina sólo imprimirá desde la izquierda de la pantalla. Así es como debe utilizarse la función; póngala en una sentencia PRINT como el argumento de TAB recordando colocar la cadena a imprimir entre paréntesis, o sea, PRINT TAB FN M(A$);A$ o PRINT TAB FN M("PERRO-");"PERRO".

(14) Para encontrar la base logarítmica 10 de un número N usando los logaritmos naturales del Spectrum:

DEF FN L(N) = LN N/LN 10
La fórmula puede utilizarse para encontrar logaritmos a otras bases, o sea, la base B:

DEF FN L(N,B) = LN N/LN B

(15) Redondeando una cantidad A a posiciones de 2 decimales:

DEF FN R(A) = INT (A * 100 + 0.5)/100

Redondeando una cantidad A a posiciones decimales D:

DEF FN R(A,D) = INT (A * (10 ↑ D) + 0.5)/(10 ↑ D).

CANALES INPUT–OUTPUT

Es posible usar ciertos comandos para suministrar OUTPUT (SALIDA) a diferentes destinos o recibir INPUT (ENTRADA) desde ciertas fuentes, o sea, pruebe INPUT #2; "Esto es #2"; #1;X.

Esto nos introduce al sistema de los canales utilizados para la entrada y salida de información en el Spectrum. Hay 19 canales posibles. Los canales –3 al –1 son utilizados por el sistema operativo, pero prácticamente no puede ser utilizado por el programador. Con el Spectrum solo (sin el módulo de expansión, microdrives, etc.), los canales 0 al +3 son implementados de la siguiente forma. Observe como sólo el canal 1 permite el INPUT normalmente:

Canal	Entrada	Salida
0	ninguno	Espacio de trabajo / pantalla inferior
1	teclado	área de edición / pantalla inferior
2	ninguno	pantalla superior
3	ninguno	impresora

Los canales 4 al 15 no se usan en el momento de escribir (a menos que hayan sido abiertos especialmente por el usuario) aunque probablemente serán utilizados por alguna opción de expansión como la RS232, microdrives, etc. Podemos hacer uso de ciertos canales para imprimir, por ejemplo, pruebe:

```
10 FOR A=0 TO 3
20 PRINT #A;"ESTO ES #";A
30 NEXT A: PAUSE 0
```

Observe como la salida del PRINT puede ir a ambas partes de la pantalla e incluso a la impresora. El PRINT no es el único comando que puede utilizarse de esta forma. Pruebe:

```
10 INPUT #2;"Entra un número "
;#1;x
```

El mensaje indicador de la cadena para el INPUT aparece en la pantalla superior (observe que no se borra a continuación) donde cualquier cosa que Vd. escriba aparece en el lugar normal. Esto es porque la tecla INPUT sólo puede venir del canal 1 determinado ya por la ROM. El OUTPUT puede ir a cualquier canal especificado. El LLIST también puede ser reemplazado por PRINT #3 y el LPRINT reemplazado por PRINT #3. Pruebe el LPRINT #2. ¿Quién necesita más el PRINT? La aplicación más obvia para esto es usar el PRINT #1 para que nos permita imprimir en esas dos líneas de la pantalla inferior. Para ver lo que sucede cuando intenta obtener un INPUT desde los canales 0, 2 o 3, pruebe INPUT #3;X.

Este ejemplo intenta obtener un INPUT desde la impresora, cosa que Vd. no puede hacer, obviamente. Hay un error del Periférico I/O de J no válido. Vd. puede abrir un canal con la sentencia OPEN#. Para hacer esto debe especificar qué periférico corresponde a ese canal. Después de # viene el número del canal, entonces una coma, después entre comillas una letra para denotar el "periférico"; R, K, S o P espacio de trabajo/pantalla inferior, área de edición-/teclado y pantalla inferior, pantalla superior e impresora respectivamente, o sea OPEN 5,"P". Lo mismo puede hacerse para cerrar un canal usando CLOSE #5, por ejemplo. Asegúrese de cerrar un canal antes de que haya sido abierto si no le acarreará graves resultados.

CONTROLAR ESOS NUMEROS

La impresión de los números a la pantalla a menudo puede ser un arte si se necesita cualquier control sobre esos números. Especialmente los programas serios necesitan que los números sean tabulados de una cierta forma. Debería asegurarse de que sólo se imprimen unas cuantas posiciones decimales, que todos los números son impresos uno debajo del otro con los puntos decimales alineados, o de que todos los números contienen la misma cantidad de dígitos.

Todo esto hace que se pida que el número se convierta en una cadena examinada después carácter por carácter para encontrar el punto decimal, la parte entera estudiada para decidir cuántos ceros a añadir, o revisar la longitud, y así sucesivamente. Veremos las rutinas para suministrar los varios tipos de formato que a menudo se usa.

(1) Redondear un número para un conjutno de posiciones decimales
Numéricamente es bastante simple redondear una cantidad (AMOUNT) a 2 posiciones decimales. Todo lo que se necesita es una fórmula como:

```
LET DOSDEC=INT (CANTIDAD*100+0.5
)/100
```

Es muy útil para los valores del dinero, ya que Vd. puede redondear hasta una peseta. Este corto programa le mostrará lo que puede hacer la rutina y algunos de sus trucos. La línea 10 genera una cantidad AMOUNT en random (aleatorio) que la línea 20 reduce a TWODEC, una copia de AMOUNT se redondea a dos posiciones decimales. Ambas se imprimen una al lado de la otra en la línea 30 para comparar.

```
  10 LET CANTIDAD=RND*100
  20 LET DOSDEC=INT (CANTIDAD*10
0+0.5)/100
  30 PRINT CANTIDAD,DOSDEC
  40 GO TO 10
```

```
34.892273           34.89
16.993713           16.99
74.623108           74.62
96.762085           96.76
57.159424           57.16
87.005615           87.01
25.434875           25.43
7.699585            7.7
77.574158           77.57
18.086243           18.09
56.561279           56.56
42.144775           42.14
60.923767           60.92
69.326782           69.33
99.543762           99.54
65.782166           65.78
33.700562           33.7
27.616882           27.62
71.348572           71.35
51.174927           51.17
38.174438           38.17
63.153076           63.15
```

La rutina funciona bastante bien, quizás un poco desordenadamente. Los números menores que 0.1 se imprimen sin ceros antes del punto decimal y los números mayores que o iguales a 0.1 y menores que 1 se imprimen con un cero antes del punto decimal. También puede haber una cantidad de dígitos de la variable después del punto decimal, es decir, el número 2 (sin ningún dígito después del punto decimal), 2.2 (1 dígito después del punto decimal) y 2.24 (que tiene dos dígitos después del punto decimal).

Si Vd. desea redondear una cantidad CANTIDAD a LUGARES (posiciones) decimales, use esta rutina. El redondeo sucede una vez se ha redondeado la cantidad:

```
LET REDONDEO=INT (CANTIDAD*(10↑L
UGARES)+0.5)/(10↑LUGARES)
```

El uso del exponente hace que la rutina sea lenta comparada con la versión anterior. Si esta última rutina se ha de usar más de una vez, trate de almacenar la exponenciación en otra variable para que las cosas vayan más rápidas:

```
LET MULT=10↑D:LET REDONDEO=INT (
CANTIDAD*MULT+0.5)/MULT
```

(2) Redondear un número fijo de posiciones decimales con ceros y/o el punto decimal añadido si es necesario.

Aquí, AMOUNT es la cantidad a imprimir en 2 posiciones decimales. Esto termina en una cadena A$, una cadena con cualquier dígito adicional añadido si es necesario.

```
   10 LET CANTIDAD=RND*100
   20 LET A$=STR$ (INT (CANTIDAD*
100+0.5) /100)
   30 IF A$(1)="." THEN LET A$="0
"+A$
   40 LET C=LEN A$-LEN STR$ INT V
AL A$
   50 LET A$=A$+".00"(C+1 TO )
   60 PRINT "£";CANTIDAD,"£";A$
   70 GO TO 10
```

```
£68.476868          £68.48
£35.800171          £35.80
£85.08606           £85.09
£81.471252          £81.47
£10.36377           £10.36
£77.384949          £77.38
£3.8955688          £3.90
£92.277527          £92.28
£20.822144          £20.82
£61.750793          £61.75
£31.352234          £31.35
£51.495361          £51.50
£62.207031          £62.21
£65.570068          £65.57
£17.793274          £17.79
£34.588623          £34.59
£94.221497          £94.22
£66.618347          £66.62
£96.414185          £96.41
£31.066895          £31.07
£30.09491           £30.09
£57.197571          £57.20
£89.866638          £89.87
£40.008545          £40.01
£0.70800781         £0.71
```

La salida de este programa es lo que puede esperar ver en una etiqueta de precios, en libras. La línea 10 determina la cantidad inicial y la línea 20 lo convierte a una cadena equivalente redondeada a dos lugares decimales. La línea 30 añade un cero al principio, si es necesario. La línea 40 busca la longitud de la parte del número sustraido de la longitud total, es decir, dígitos después del punto decimal, si los hay. Esto se usa para determinar cuantos ceros/puntos decimales se han de añadir a la línea 50. La línea 60 imprime ambas versiones juntas para comparar.

(3) Alinear los puntos decimales

Donde Vd. tiene cartas de números, un entendimiento visual se obtiene si los números están alineados de tal forma que los puntos decimales van uno detrás de otro.

```
10 LET CANTIDAD=RND*1000
20 PRINT CANTIDAD
30 GO TO 10
```

```
1.1291504
85.81543
437.19482
790.25269
269.1803
169.34631
201.68904
142.57813
694.33594
75.531006
665.8783
941.25366
594.08569
556.88477
766.86096
514.63154
612.91504
969.07043
680.31311
23.834229
788.68103
151.30615
```

Para alinear los puntos decimales, todo lo que se necesita es cambiar un poco la línea 20:

```
   10 LET CANTIDAD=RND*1000
   20 PRINT TAB 15-LEN STR$ INT C
ANTIDAD;CANTIDAD
   30 GO TO 10
```

```
169.93713
746.23108
967.62085
571.59424
870.05615
254.34875
 76.99585
775.74158
180.86243
565.61279
421.44775
609.23767
693.25782
995.43762
657.82166
337.00562
276.16882
```

```
713.48572
511.74927
381.74438
631.53076
365.21912
```

¿Ve como queda todo más ordenado ahora?. Si desea alinear los puntos decimales de los números formateados a 2 posiciones decimales contenidos en una cadena, no siempre puede usar el método de arriba porque la cadena puede ser afectada convergiendo con el VAL. Sin embargo, Vd. sabe cuántos lugares decimales habrán, así pues:

```
20 PRINT TAB 15-LEN A$;A$
```

Observe que algo sucede cuando los números llegan a ser tan largos que el STR$ comienza usando una notación científica. Naturalmente, a menos que Vd. esté tratando con cantidades mayores de cien millones, no debe preocuparse.

(4) Sobreescribiendo columnas de números

Cuando tabula los números en columnas, es normal sobreimprimir números unos encima de los otros para poner al día los valores. Esto hace correr el riesgo de un valor que tenga un número diferente de dígitos de otro,dejándolos en la pantalla. Como un simple ejemplo, haga:

```
10 FOR A=110 TO 0 STEP -1
20 PRINT AT 0,0;A
30 NEXT A
```

Los números en la parte superior de la pantalla comienzan como números de tres dígitos, pero una vez los números llegan a ser números de 2 dígitos un cero permanece a la derecha de la columna. Esto podría evitarse imprimiendo unos cuantos espacios después del número. Naturalmente, si Vd. tiene gráficos/texto/otra columna de números naturalmente también serán borrados.

Use esta rutina para imprimir sólo los espacios suficientes para alcanzar el número más largo (en términos de dígitos). La línea 20 debería tener en las comillas de impresión un espacio menos que el número máximo de dígitos a imprimir. El número después del TO debería tener esa cantidad máxima de dígitos.

```
  10 FOR A=110 TO 0 STEP -1
  20 PRINT AT 0,0;A;"  "( TO 3-L
EN STR$ A)
  30 NEXT A
```

RUTINAS DE LA ROM

Hay un número de rutinas útiles en la ROM de 16K que pueden utilizarse en programas en código de máquina escritos por el usuario. Esta lista detalla algunas de ellas, pero de ninguna forma es exhaustiva ya que si no se hubiera convertido en el tema principal de este libro. Todas las direcciones que se relacionan están en decimal salvo que se indique otra cosa.

Rutina 16 PRINT. El contenido del registro A es enviado al actual canal OUTPUT. Si el canal es otro, menos el que Vd. quiere hacer un PRINT, use la rutina en 5633 con el registro A que contiene el número del canal a abrir (normalmente la pantalla superior, canal 2). Vd. puede usar la rutina PRINT para hacer un OUTPUT de los caracteres de control igual que los caracteres normales a imprimir.

949 BEEP. Esto hará sonar un ruido en el altavoz y zócalos de la cinta como el comando BEEP del BASIC. En la entrada a esta rutina, el par de registros HL deberían contener la frecuencia (los valores más inferiores dan una frecuencia más alta) y el par de registros DE deberían contener la duración de la nota (los valores más bajos dan duraciones más cortas). Observe que la frecuencia afecta a la duración de la nota de forma que si Vd. doblara la frecuencia y tuviera el mismo valor de duración, la nota no debería tener la misma longitud. Llamando a esta rutina repetidamente con una breve duración del BEEP de diferentes frecuencias, simularía diferentes tonos, incluso el control del envolvente, algo que no es posible con el BASIC.

3435 CLS. El uso de esta rutina es el mismo que el comando CLS del BASIC.

3582 Deslizar la pantalla. El USR 3582 o el CALL 3582 deslizarán la pantalla hacia arriba una línea de caracteres sin que afecte a la posición PRINT o a la variable del sistema SCR–CT. Es muy útil para los juegos donde siempre se encuentra la acción del deslizamiento sin tener que usar algo así como:

POKE 23692,255: PRINT AT 21,31;”

el cual fuerza un deslizamiento de la pantalla y suprime la indicación ”scroll?”

3742 Dirección del comienzo de la línea de la pantalla. Si el registro A se carga con el número de la hilera PRINT (0 a 23 pantalla abajo) el valor de HL como respuesta a esta rutina será la dirección de la línea de puntos en la parte superior de la pantalla en el primer carácter en esa línea.

3756 Igual que el comando COPY del BASIC. La parte interesante es que después de desactivar las interrupciones comienza a especificar cuántas líneas de la parte superior de la pantalla serán copiadas a la impresora. Así pues, Vd. puede reemplazar esta parte con su propia rutina para especificar cuántas líneas de la parte superior de la pantalla se han de copiar a la impresora. Este número debería estar en el registrador B. La rutina es ésta:

DI	:	desactivar las interrupciones
LD B, número		cuántas líneas a copiar
CALL 3759		COPIELO a la impresora

3789 LPRINT. Imprime lo que se encuentre en el buffer de la impresora a la impresora del ZX y reinicializa todo lo que concierne al buffer de la impresora.

6683 Imprime el contenido del par de registros BC como un número decimal. Del 0 al 9999 son los valores permitidos ya que se usan normalmente para imprimir los números de líneas.

7997 PAUSE El registro BC maneja el tiempo de espera en unos 50avos de segundo en Europa. Como el comando PAUSE del BASIC, este rato de espera termina cuando el tiempo de espera termina o se pulsa una tecla. Como el PAUSE0, si BC contiene el cero para la entrada, el registro C contiene la coordenada X y el registro B contiene la coordenada Y. El número en el registro A denota qué bit de la dirección en el HL corresponde al eje X, Y de la pantalla.

8252 PRINT bytes. Esta rutina imprimirá al actual canal de salida una cadena que comienza en la dirección que maneja el par de registros DE de la longitud almacenada en el par de registros BC.

8855 color de los lados. Para cambiar el color de los rincones ponga el número del color en el registro A la llamada 8855.

8874 Esta rutina le dirá qué bit o qué dirección en la pantalla corresponde al eje de la pantalla. En la entrada el registro C contiene la coordenada X y el registro B contiene la coordenada Y. El número en el registro A denota qué bit de la dirección HL corresponde al eje X, Y de la pantalla.

8927 rutina PLOT. El registro B debería contener la coordenada Y, el registro C la coordenada X en la entrada. Corresponde al PLOT X, Y.

9402 DRAW líneas rectas. Cuatro registros necesitan determinarse de la siguiente forma, antes de la llamada de las rutinas. Cargue el registro B con el offset Y (valor absoluto) y cargue el registro C con el offset X (valor absoluto). Entonces, los registros D y E tienen que contener un 1 para offset-positivo o 255 para offset-negativo (el SGN de los offsets). El registro D denotará el signo del offset X y el registro E el signo del offset Y.

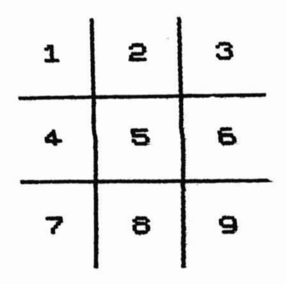
TRES EN RAYA

Este programa juega el juego de las tres en raya,en el cual el objeto es para Vd. o el computador tomar diferentes turnos para obtener tres Os o tres Xs respectivamente en la hilera de arriba, a través o en diagonal. Al contrario de muchos programas para jugar este juego, al computador se le puede ganar, pero no siempre. El programa juega defendiéndose, así que la mayoría de los juegos acaban con un dibujo.

Cuando Vd. ejecuta el programa,éste le pregunta si Vd. quiere comenzar primero. Conteste s si quiere empezar primero, o con una n si no. Asegúrese de que la tecla CAPS LOCK no está activada, pues el programa sólo reconoce letras en minúsculas. Si Vd. añade la línea:

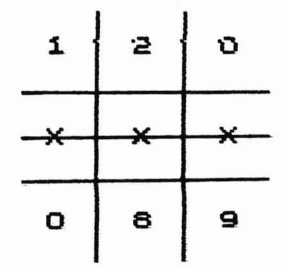
15 POKE 23658,0

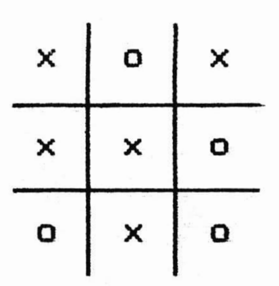
esto hará que el propio computador desactive el CAPS LOCK. El programa usa el INKEY$ para detectar las respuestas del teclado, así que Vd. no tiene que pulsar ENTER excepto cuando ejecute el programa. El tablero queda numerado de la forma siguiente:



Para hacer el movimiento, pulse la tecla con el mismo número que el cuadrado donde Vd. quiere colocar una O; el computador es siempre la X. Una vez haya hecho una línea ganadora, el computador dibuja una línea a través de esa línea y saca un mensaje destellante en la pantalla indicando quién ha ganado. A continuación verá algunos ejemplos:

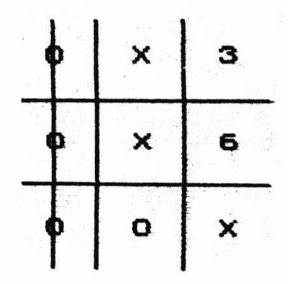
¡HE GANADO!





EMPATE

HAS GANADO



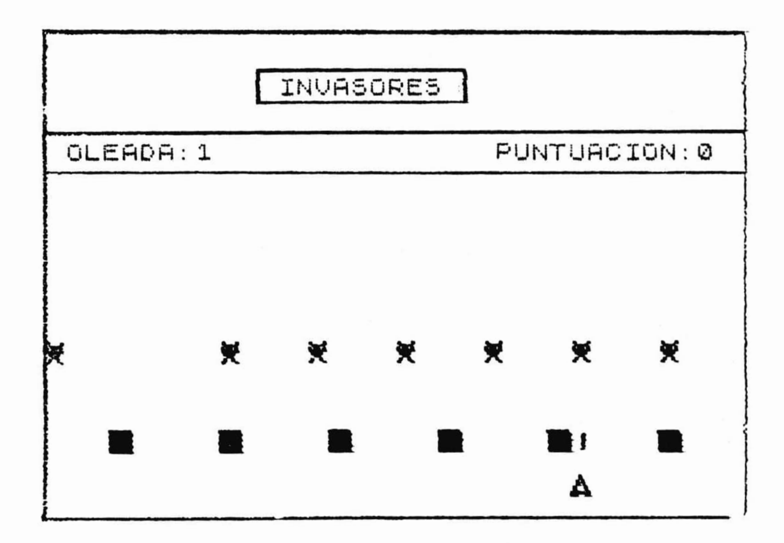
```
   1 REM TRES EN RAYA
   2 REM por Dilwyn Jones.
   3 POKE 23658,8
  10 PRINT AT 0,0;"SALES TU (S/N
)?"
  20 LET A$=INKEY$: IF A$<>"S" A
ND A$<>"N" THEN GO TO 20
  30 IF A$="S" THEN GO SUB 1000:
 GO SUB 1000
  40 IF A$="N" THEN GO SUB 1000:
 GO TO 300
  90 GO SUB 1000: REM INICIAR TU
 MOVIMIENTO
 110 PRINT AT 18,12;"TU JUEGAS"
 120 LET A=CODE INKEY$-48: IF A<
1 OR A>9 THEN GO TO 120
 130 IF B$(A)<>" " THEN GO TO 12
0
 140 LET B$(A)="O"
 150 PRINT AT Y(A),X(A);B$(A)
 160 LET A$="OOO"
 170 GO TO 600: REM GANAS?
 300 REM JUGADA SPECTRUM
 310 PRINT AT 18,12;"MI JUGADA"
 315 LET A$="XXX"
 320 IF B$(5)=" " THEN LET F=5:
GO TO 560: REM MITAD TABLERO
 330 FOR B=1 TO 2
 340 IF B=1 THEN LET C$="XX"
 350 IF B=2 THEN LET C$="OO"
 360 RESTORE 420
 370 FOR A=1 TO 70 STEP 3
 380 READ D,E,F
 390 IF B$(D)+B$(E)=C$ AND B$(F)
=" " THEN GO TO 560: REM COMBINA
CION GANADOR-HACER ALGO
 400 NEXT A
 410 NEXT B
 420 DATA 1,5,9,1,9,5,5,9,1,3,5
 430 DATA 7,5,7,3,3,7,5,1,2,3,1
 440 DATA 3,2,2,3,1,4,5,6,5,6,4
 450 DATA 4,6,5,7,8,9,7,9,8,9,6
 460 DATA 7,1,4,7,4,7,1,1,7,4,2
 470 DATA 5,8,2,8,5,5,8,2,3,6,9
 480 DATA 3,9,6,6,9,3
 500 REM MOVIMIENTO AL  AZAR
 510 LET C$=""
 520 FOR A=1 TO 9
 530 IF B$(A)=" " THEN LET C$=C$
+STR$ A
 540 NEXT A
 550 LET F=VAL C$(INT (RND*LEN C
$)+1)
 560 LET B$(F)="X"
 570 PRINT AT Y(F),X(F);B$(F)
 580 LET A$="XXX"
 600 REM COMPROBAR QUIEN GANA
 610 RESTORE 700
 620 FOR A=1 TO 8
 630 READ D,E,F
```

```
 640 IF B$(D)+B$(E)+B$(F)=A$ THE
N GO TO 900
 650 NEXT A
 660 LET X=X+1
 670 IF X=9 THEN GO TO 800
 680 IF A$="XXX" THEN GO TO 100
 690 IF A$="OOO" THEN GO TO 300
 700 DATA 1,2,3,4,5,6,7,8,9,1,4
 710 DATA 7,2,5,8,3,6,9,1,5,9,3
 720 DATA 5,7
 730 STOP
 800 REM EMPATE
 805 BEEP 3,-10
 810 PRINT AT 18,12;"          ";
AT 21,11; FLASH 1;"EMPATADOS!"
 820 STOP
 900 REM GANAR
 905 INK 4
 915 IF A=1 THEN PLOT 80,131: DR
AW 90,0
 920 IF A=2 THEN PLOT 80,99: DRA
W 90,0
 925 IF A=3 THEN PLOT 80,67: DRA
W 90,0
 930 IF A=4 THEN PLOT 91,143: DR
AW 0,-90
 935 IF A=5 THEN PLOT 123,143: D
RAW 0,-90
 940 IF A=6 THEN PLOT 155,143: D
RAW 0,-90
 945 IF A=7 THEN PLOT 77,143: DR
AW 90,-90
 950 IF A=8 THEN PLOT 169,143: D
RAW -90,-90
 955 INK 0
 957 PRINT AT 18,12;"          "
 960 IF A$="OOO" THEN FOR X=65 T
O 10 STEP -1: BEEP 0.08,X: NEXT
X: PRINT AT 2,12; FLASH 1;"HAS G
ANADO!!"
 965 IF A$="XXX" THEN FOR Z=10 T
O 65: BEEP 0.08,Z: NEXT Z: PRINT
 AT 2,13; FLASH 1;"HE GANADO"
 975 STOP
1000 REM INICIO PANTALLA
1010 BORDER 6: PAPER 6: CLS
1020 PRINT AT 5,11;"1   2   3";A
T 9,11;"4   5   6";AT 13,11;"7
 8   9"
1025 INK 4
1030 PLOT 80,115: DRAW 90,0
1040 PLOT 80,83: DRAW 90,0
1050 PLOT 107,143: DRAW 0,-90
1060 PLOT 139,143: DRAW 0,-90
1070 INK 0
2000 REM INICIAR VARIABLES
2010 DIM B$(9): REM TABLERO
2020 LET X=0: REM CONTADOR DE MO
VIMIENTOS
2030 DIM X(9): REM X COORDENADAS
```

```
2040 DIM Y(9): REM Y COORDENADAS
2050 RESTORE 2090
2060 FOR A=1 TO 9
2070 READ X(A),Y(A)
2080 NEXT A
2090 DATA 11,5,15,5,19,5,11,9,15
2100 DATA 9,19,9,11,13,15,13,19
2110 DATA 13
2120 RETURN
```

INVASORES

Un ejército de extraterrestres descienden. Vd. debe prevenir que no aterrizen a toda costa o destruirán la Tierra. Debe enfrentarse a ellos solamente armado con su pistola láser y protegido por unas cuantas defensas. Los extraterrestres se le acercan en formación de batalla disparándole con sus misiles con bastante puntería. Tiene que evitarlos a toda costa o esconderse detras de sus defensas, pero tenga en cuenta que después de dispararles se evaporizan. Apunte al extraterrestre y vaporícelo disparándole con su cañón láser; verá al alienígena que cambia de color al recibir el disparo. El láser hará lo mismo con sus defensas así que procure no destruirlas!. Vd. no puede ganar – le cogerán pero trate de hacer una puntuación más alta que nadie. Use el 5 (cursor izquierda) para moverse a la izquierda, el 8 (cursor derecha) para moverse a la derecha y el 7 (cursor arriba) para disparar su cañón láser contra los extraterrestres. Si alguno de ellos desaparece en el eje de la pantalla, reaparecerá en lado opuesto, así que no se lo permita.



```
   1 REM INVASORES
         By Dilwyn Jones
  10 GO SUB 9000
  20 GO SUB 9500
  30 FOR D=1 TO 7
  34 LET C$="  ■    ■    ■    ■
 ■    ■   "
  35 PRINT INK 2;AT 18,0;C$
  40 LET A$="Ж    Ж    Ж    Ж    Ж
Ж    Ж   "
  70 PRINT AT 5,10;D
  80 FOR B=D+9 TO 19 STEP 2
  85 IF B>=16 THEN LET C$=B$: PR
INT AT 18,0;C$
  90 FOR E=1 TO 3
 100 LET A$=A$(32)+A$( TO 31)
 110 FOR A=0 TO DIF
 111 IF A$(X+1)="Ж" AND RND<.05
AND NOT BOMBY THEN LET BOMBY=B:
LET BOMBX=X
 112 IF BOMBY>18 AND BOMBX=X THE
N GO TO 3500
 113 IF BOMBY THEN GO SUB 3000
 120 LET X=X+(INKEY$="8" AND X<3
1)-(INKEY$="5" AND X>0)
 130 PRINT AT B,0; INK 4;A$; INK
 3;AT 20,C;" " AND C<>X;AT 20,X;
"Á"
 140 IF A$=B$ THEN GO TO 240
 150 LET C=X
 160 IF INKEY$="7" THEN GO SUB 4
000
 170 NEXT A
 180 NEXT E
 190 PRINT AT B,0;B$
 200 NEXT B
 210 PRINT PAPER 2; FLASH 1;AT 7
,5;"TE HAN INVADIDO"
 230 STOP
 250 NEXT D
 260 LET DIF=DIF-1: IF DIF<5 THE
N LET DIF=20
 300 GO TO 30
 980 DRAW 255,0
1000 REM HACER RUIDO
1020 BEEP .01,30: BEEP .01,40
1080 RETURN
1100 RETURN
2000 REM MAS RUIDO
2010 FOR A=0 TO 21 STEP 2
2020 BEEP 0.01,(40-(A/2))
2030 NEXT A
2040 RETURN
3000 REM BOMBAS
3010 LET BOMBY=BOMBY+1
3015 IF BOMBY=21 THEN PRINT AT B
OMBY-1,BOMBX;" ": LET BOMBY=0: R
ETURN
3020 IF BOMBY=18 AND C$(BOMBX+1)
="■" THEN GO SUB 3100
```

```
3030 IF BOMBY-1>B THEN PRINT AT
BOMBY-1,BOMBX;" "
3040 IF BOMBY THEN PRINT AT BOMB
Y,BOMBX;"|"
3050 RETURN
3100 REM GOLPEAR BLINDAJE
3110 LET C$(BOMBX+1)=" "
3115 IF BOMBY-1<>B THEN PRINT AT
 BOMBY-1,BOMBX;" "
3120 LET BOMBY=0
3130 PRINT AT 18,BOMBX;" "
3135 GO SUB 1000
3140 RETURN
3500 REM DISPARO
3510 PRINT AT BOMBY,BOMBX;" ";AT
 20,X; FLASH 1; INK 7;"A": GO SU
B 2000: PRINT AT 21,0;"DESTRUIDO
..!@#@#@#@#!"
3520 STOP
4000 REM DESTRUIR BLINDAJE
4010 IF C$(X+1)=" " THEN GO TO 5
000
4020 INK 9
4030 PLOT X*8+3,16: DRAW 0,8
4040 LET C$(X+1)=" "
4050 PLOT OVER 1;X*8+3,16: DRAW
OVER 1;0,8
4055 GO SUB 1000
4060 PRINT AT 18,X;" "
4070 INK 6
5000 REM FUEGO LASER
5005 IF A$(X+1)="X" THEN LET PUN
TOS=PUNTOS+1: IF PUNTOS>999 THEN
 LET PUNTOS=0
5010 INK 9
5020 PLOT X*8+3,16
5030 DRAW 0,(19-B)*8
5040 OVER 1
5050 PLOT X*8+3,16
5060 DRAW 0,(19-B)*8
5070 INK 6
5080 OVER 0
5085 GO SUB 1000
5090 LET A$(X+1)=" "
5095 PRINT AT 5,29; INK 6;PUNTOS
5100 RETURN
9000 REM INICIALIZACION  X A
9010 RESTORE 9080
9020 FOR C=0 TO 7
9030 READ A,B
9040 POKE USR "A"+C,A
9050 POKE USR "B"+C,B
9060 NEXT C
9080 DATA 66,16,60,16,90,40,126,
40,60,68,24,68,36,186,66,198
9100 RETURN
9500 REM INICIO
9510 PAPER 0
9520 BORDER 4
9530 CLS
9540 INK 3
```

```
9550 PLOT 0,139
9560 DRAW 255,0
9570 PLOT 0,124
9590 INK 6
9600 PRINT FLASH 1;AT 2,10;"INVA
SORES"
9610 PLOT 78,161
9620 DRAW 75,0
9630 DRAW 0,-11
9640 DRAW -75,0
9650 DRAW 0,11
9660 PRINT AT 5,1;"Pantalla:0";T
AB 22;"Puntos=0"
9670 DIM A$(32)
9680 DIM B$(32)
9690 DIM C$(32)
9695 LET BOMBY=0: LET BOMBX=0
9696 LET X=INT (RND*32): LET C=X
9700 LET PUNTOS=0
9705 LET DIF=20
9710 RETURN
```

El programa necesita 12K incluyendo la pantalla, variables, el programa y los gráficos definidos por el usuario para poderse ejecutar, así que es suficiente para el Spectrum de 16K. Es un juego escrito en BASIC, que usa varios colores, gráficos de alta resolución, gráficos definidos por el usuario y sonido para excitar la acción rápida. Tienen una facilidad de puntuación , lo cual suministra un elevado sentido de competición. El uso extensivo está compuesto de cadenas para un manejo de datos más rápido y los comandos PLOT, DRAW y OVER se utilizan para dibujar y borrar líneas. El programa en sí está contenido en las líneas 30 a la 300 y el resto del programa son principalmente subrutinas para llevar a cabo varias funciones. Todas están remarcadas con REMs para poder identificarlas. Todo lo que esto necesita añadir es observar los caracteres gráficos en cada línea:

34: gráficos SHIFT 8
40: gráficos A
111: gráficos A
130: gráficos B
3020: gráficos SHIFT B
3510: gráficos B
5005: gráficos A
9000: gráficos A seguidos por gráficos B

Si Vd. intenta cambiar el programa, observe que ha sido escrito para ser rápido, así pues, procure no hacerlo más lento. Los comandos de sonido en la subrutina 1000 deben mantenerse muy cortos; unos 0.95 segundos máximo. Haga lo que Vd. quiera hacer con la otra subrutina de sonido, ya que sólo sucede al final del programa cuando Vd. ha sido derrotado, ¡es más larga para hacerle sentir más humillado!

SUPER SONIDOS

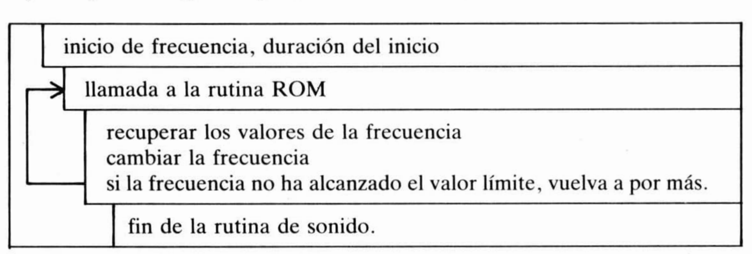
En BASIC las facilidades de sonido del Spectrum son limitadas para crear una sola nota de una duración fija y frecuencia como:

BEEP duración, frecuencia

La podemos extender un poco tocando varias notas cortas y muy deprisa una después de la otra, por ejemplo:

```
10 FOR A=0 TO 21
20 BEEP 0.01,(40-(A/2))
30 NEXT A
```

Sin embargo, debemos llegar a la conclusión de que para producir cualquier sonido más complejo, el BASIC es incapaz de ello. Incluso cuando estamos en lenguaje de máquina, todavía estamos limitados a controlar la frecuencia y la duración de un sonido. Sin embargo, lo que sí puede hacer el código de máquina es incrementar el proceso de tal forma que podríamos oír una secuencia de notas, una después de la otra, tan de prisa que pasaría a convertirse en una nota larga. Y si nosotros miramos de arreglar un poco estas notas, podríamos conseguir notas normalmente imposibles (algo similar a los discos pop). La forma de hacer esto es usando la rutina de la ROM que normalmente maneja el comando BEEP repetidamente llamando con muy pocas notas el cambio de frecuencias. Aquí hay un diagrama para ilustrarlo:



Aquí hay un programa en código de máquina que nos permitirá hacer esto. Si Vd. no entiende el código de máquina o lenguaje assembler, se le darán el máximo de instrucciones para que Vd. pueda ejecutar las rutinas en su Spectrum de 16K o de 48K. Estos sonidos pueden utilizarse en cualquier programa si Vd. procura añadirle estas rutinas:

HEXA-DECIMAL	**DECIMAL**	**ASSEMBLER**	**NOTAS**
Ø6 ØA	6,1Ø	LD B,1Ø	;contador de repeticiones
C5	197	COUNT; PUSH BC	;salvar contador en stack
21 ØØ ØØ	33,Ø,Ø	LDHL,Ø	;inicio de frecuencia
21 64 ØØ	17,1ØØ,Ø	LOOP ; LD DE,1ØØ	;duración de un sonido

E5	229	PUSH HL	;salvar el valor de la frecuencia
CD B5 Ø3	2Ø5,181,3	CALL949	;llamada a la rutina ROM BEEP
Ø1 14 ØØ	1,2Ø,Ø	LD BC,2Ø	;salto de frecuencia
11 64 ØØ	17,1ØØ,Ø	LD DE,1ØØ	;valor límite de frecuencia
E1	225	POP HL	;recuperar frecuencia del stack
C6 ØØ	198,Ø	ADD A,Ø	;recuperar flag de acarreo
ED 4A	237,74	ADC HL,BC	;nuevo valor de frecuencia
E5	229	PUSH HL	;salvar nueva frecuencia en el stack
C6 ØØ	198,Ø	ADD A,Ø	;recuperar flag de acarreo
ED 52	237,82	SBC HL,DE	;comprobar frecuencia límite
E1	225	POP HL	;recuperar valor de frecuencia
38 E6	56,23Ø	JR C,LOOP (JR C, -26)	;¿límite de frecuencia alcanzado?
C1	193	POP BC	;recuperar contador de iteraciones
1Ø DF	16,223	DJ NZ,COUNT (DJ NZ, -33)	;¿más repeticiones?
C9	2Ø1	RET	;retornar al BASIC

Saltos relativos le han sido mostrados en assembler ya que ambos saltan a niveles de saltos de claridad y relativos (menos) para propósitos prácticos. Los valores que se dan a los números cargados en los registros son ejemplos. Los marcadores a lo largo del assembler nos indicarán lo que allí sucede.

Por encima, la rutina anterior, de 36 bytes, toma una nota de frecuencia HL y duración DE y la hace sonar repetidamente con frecuencia descendente hasta alcanzar el valor límite, repitiéndose el proceso el número de veces especificado. Observe que con el uso de la rutina ROM BEEP, de esta manera, la duración depende de la frecuencia (se debe de pensar en número de ciclos en vez de en duración). Una frecuencia más elevada conlleva un período más corto.

Tanto la frecuencia como la duración pueden comenzar con un valor de 1, que corresponde a la frecuencia más alta y la duración más corta respectivamente. Cuando se aumenta el valor de ambos el resultado será un tono bajo de una duración excesiva y, normalmente, poco útil. Lo que haré es darle un cargador de código máquina en valores decimales,el cual depositará dichos valores, contenidos en líneas DATA mediante POKEs por encima de la RAMTOP. Se necesitan reservar 36 bytes para una de estas rutinas. Sugiero para los usarios de la máquina de 16K un CLEAR 32563, de modo que la rutina comenzará en la 32564. En máquinas de 48K le sugiero que la primera línea del programa sea CLEAR 65331 para que la rutina comience en la 65332. Los dos listados siguientes son para las máquinas de 48K y 16K respectivamente:

```
   1 REM CARGADOR DE SONIDOS 48K
  10 CLEAR 65331
  20 LET DIRECCIONS=65332: LET F
IN=1000
  30 READ BYTE: IF BYTE=FIN THEN
 STOP
  40 IF BYTE>255 THEN STOP
  50 POKE DIRECCIONS,BYTE
  60 LET DIRECCIONS=DIRECCIONS+1
  70 GO TO 30
```

```
  1 REM CARGADOR SONIDOS 16K
 10 CLEAR 32563
 20 LET DIRECCIONS=32564: LET F
IN=1000
 30 READ BYTE: IF BYTE=FIN THEN
 STOP
 40 IF BYTE>255 THEN STOP
 50 POKE DIRECCIONS,BYTE
 60 LET DIRECCIONS=DIRECCIONS+1
 70 GO TO 30
```

La línea 10 coloca la nueva RAMTOP en lugar en que irá el código de máquina. Este valor deberá de ajustarse en caso de que se tenga que poner más código de máquina (por ejemplo, si se combina más de una rutina, cada una de las cuales necesitaría 36 bytes adicionales). Para determinar este valor vea cuál es el número en la variable del sistema RAMTOP (normalmente es 32599 para el Spectrum de 16K y 65367 para el de 48K) a continuación reste el número de bytes de la rutina en lenguaje de máquina (por ejemplo, en un Spectrum de 16K tendríamos 32599–36=32563), de esta forma la rutina arranca en la posición inmediatamente posterior al nuevo valor RAMTOP. En la línea 20, ADDRESS es el lugar donde comienza la rutina, a continuación de RAMTOP, tal como se ha dicho. La variable END se usa como marcador de fin de datos en la lista DATA. Ello quiere decir que los números a POKEar deben tener un valor entre 0 y 255, de modo que al leer un valor mayor que 255, se asumirá que se ha llegado al final de la lista. Se podría hacer simplemente poniendo cualquier número pero la asignación de una variable da una indicación más clara de su utilización (observe que no todos los computadores pueden leer el contenido de un nombre de variable colocado en una lista DATA). Ello se hace en las líneas 30 y 40. La línea 50 coloca los bytes del código de máquina en su lugar de la memoria, la línea 60 incrementa la dirección del siguiente byte.

A continuación veremos algunos ejemplos de los sonidos que se pueden generar. Los listados corresponden a sentencias DATA que se deben de utilizar con el programa cargador. Todos contienen 36 bytes del mismo código pero con diferentes frecuencias, repeticiones, etc. Teclee las sentencias DATA que Vd. haya elegido y ejecute el programa cargador. El programa debe de finalizar con un 9 STOP, 30:3. Si no sucede así y Vd. no ha hecho ningún cambio en el programa, es que hay un error en algún lugar. Grabe el programa en cinta para evitar perderlo mientras se ejecuta debido a algún error. Cuando ya esté listo para escuchar los sonidos, una vez el código de máquina esté listo (ésta es la mejor manera de saber si todo funciona tras haber salvado el programa en cinta), se pueden usar los siguientes comandos para ejecutar el código de máquina:

LET A=USR 32564 (Spectrum de 16K)

LET A=USR 56332 (Spectrum de 48K)

Naturalmente, Vd. podría haber utilizado algo como RANDOMIZE USR 65332, pero ello puede afectar a la aleatoriedad de los números generados si, como es de suponer, estos sonidos se van a utilizar en juegos. Como cosa interesante, donde Vd. vea una llamada a código de máquina utilizando RANDOMIZE y en el programa se usen números aleatorios, podría utilizarse RANDOMIZE (0*USR nnnnn) (lo cual equivale a un RANDOMIZE 0). Volviendo a nuestro tema principal, he aquí algunos listados con sentencias DATAS para obtener sonidos:

1000 LET A=(PEEK 23637+256*PEEK 23638+5)
2000 REM ØØØØØ ---- Nº BYTES CODIGO MAQUINA - Ø
3000 FOR B=(A+5) TO (A+5+Nº BYTES):READ C: POKE B,C:NEXT B

```
  98 REM BOMBAS CAYENDO
  99 REM LISTA DATAS SONIDO
 100 DATA 6,1,197,33,0,0,17,1
 110 DATA 0,229,205,181,3,1,20,0
 120 DATA 17,0,12,225,198,0,237
 130 DATA 74,229,198,0,237,82
 140 DATA 225,56,230,193,16,223
 150 DATA 201,FIN
```

```
  98 REM FUEGO DE LASER
  99 REM LISTA DATOS SONIDO
 100 DATA 6,1,197,33,0,0,17,1
 110 DATA 0,229,205,181,3,1,1,0
 120 DATA 17,100,1,225,196,0,237
82
 130 DATA 74,229,198,0,237,82
 140 DATA 225,56,230,193,16,223
 150 DATA 201,FIN
```

```
  98 REM FUEGO LASER REPETIDO
  99 REM LISTA DATOS SONIDO
 100 DATA 6,10,197,33,0,0,17,1
 110 DATA 0,229,205,181,3,1,1,0
 120 DATA 17,100,1,225,198,0,237
 130 DATA 74,229,198,0,237,82
 140 DATA 225,56,230,193,16,223
 150 DATA 201,FIN
```

```
  98 REM RASCAR
  99 REM LISTA DATAS SONIDO
 100 DATA 6,1,197,33,0,10,17,1
 110 DATA 0,229,205,181,3,1,100,
0
 120 DATA 17,0,30,225,198,0,237
 130 DATA 74,229,198,0,237,82
 140 DATA 225,56,230,193,16,223
 150 DATA 201,FIN
```

```
  98 REM MAQUINA EXTRATERRESTRES
S
  99 REM LISTA DATOS SONIDO
 100 DATA 6,20,197,33,0,4,17,1
 110 DATA 0,229,205,181,3,1,50,0
 120 DATA 17,0,6,225,198,0,237
 130 DATA 74,229,198,0,237,82
 140 DATA 225,56,230,193,16,223
 150 DATA 201,FIN
```

```
  98 REM SONIDOS MISTERIOSOS
  99 REM LISTA DATOS SONIDO
 100 DATA 6,5,197,33,0,10,17,10
 110 DATA 0,229,205,181,3,1,0,2
 120 DATA 17,0,19,225,198,0,237
 130 DATA 74,229,198,0,237,82
 140 DATA 225,56,230,193,16,223
 150 DATA 201,END
```

```
  98 REM ALARMA
  99 REM LISTA DATOS SONIDO
 100 DATA 6,10,197,33,0,0,17,100
 110 DATA 0,229,205,181,3,1,0,1
 120 DATA 17,0,3,225,198,0,237
 130 DATA 74,229,198,0,237,82
 140 DATA 225,56,230,193,16,223
 150 DATA 201,FIN
```

```
  98 REM  DESTELLO DEL LASER
  99 REM   LISTA DATOS SONIDO
 100 DATA 6,25,197,33,0,0,17,6
 110 DATA 0,229,205,181,3,1,50,0
 120 DATA 17,0,1,225,198,0,237
 130 DATA 74,229,198,0,237,82
 140 DATA 225,56,230,193,16,223
 150 DATA 201,END
```

```
  98 REM SINTETIZADOR 'PEEOW'
  99 REM LISTA DATOS SONIDO
 100 DATA 6,1,197,33,0,0,17,5
 110 DATA 0,229,205,181,3,1,1,0
 120 DATA 17,0,1,225,198,0,237
 130 DATA 74,229,198,0,237,82
 140 DATA 225,56,230,193,16,223
 150 DATA 201,FIN
```

```
98 REM PAJARO
 99 REM LISTA DATOS SONIDO
100 DATA 6,14,197,33,0,0,17,40
110 DATA 0,229,205,181,3,1,25,0
120 DATA 17,240,0,225,198,0,237
130 DATA 74,229,198,0,237,82
140 DATA 225,56,230,193,16,223
150 DATA 201,FIN
```

Después de estar acostumbrados a utilizar la sentencia del BASIC BEEP, los sonidos que se obtienen mediante este procedimiento nos producirán una feliz sorpresa. Lo único que esta rutina no puede hacer es generar un tono de frecuencia ascendente. Normalmente ello no es problema ya que el sonido se repite tan rápidamente que no se nota. Si Vd. desea específicamente una nota de frecuencia ascendente, he aquí otra versión de la misma rutina que lo consigue:

HEXA-DECIMAL	DECIMAL	ASSEMBLER	NOTAS
Ø6 Ø1	6,1	LD B,1	;contador de repeticiones
C5	197	REPT; PUSH BC	;salvar contador
21 E8 Ø3	33,232,3	LD HL,1ØØØ	;inicio de frecuencia
11 Ø1 ØØ	17,1,Ø	LOOP;LD DE,1	;duración de un sonido
E5	229	PUSH HL	;salvar la frecuencia
CD B5 Ø3	2Ø5,181,3	CALL949	;llamada al BEEP en la ROM
E1	225	POP HL	;recuperar frecuencia
Ø1 Ø1 ØØ	1,1,Ø	LD BC,1	;cambio de frecuencia
C6 ØØ	198,Ø	ADD A,Ø	;recuperar flag de acarreo
ED 42	237,66	SBC HL,BC	;nueva frecuencia
3Ø EF	48,239	JR NC,LOOP (JR NC, -17)	;¿más de ese sonido?
C1	193	POP BC	;recuperar el contador de iteración
1Ø E8	16,232	DJ NZ,REPT (DJ NZ, -24)	;¿más repeticiones?
C9	2Ø1	RET	;retornar al BASIC

Esta rutina sólo tiene 27 bytes de longitud y carece de la característica "frecuencia límite", significando ello que sea cual sea la frecuencia con la que arranque el tono, siempre finaliza en la más alta. Esta rutina se puede incorporar en sentencias DATA de la misma forma que las anteriores. Aunque se desperdiciarán unos cuantos bytes, cosa que si le preocupa, se puede solucionar cambiando la sentencia CLEAR en la línea 10 del programa cargador. No obstante, dejándolo tal como está tiene la ventaja de que el cargador es estándard, sirviendo para ambos tipos de rutina. Si desea que esta rutina tenga la misma longitud que la anterior, se pueden rellenar con ceros (NOP) hasta el final de DATA.

```
 98 REM TONO ASCENEDENTE
 99 REM LISTA DATOS SONIDO
100 DATA 6,1,197,33,232,3,17,1
110 DATA 0,229,205,181,3,225,1
120 DATA 1,0,198,0,237,66,48
130 DATA 239,193,16,232,201,FIN
```

```
 98 REM TONO ASCENDENTE RAPIDO
 99 REM LISTA DATAS SONIDO
100 DATA 6,1,197,33,232,2,17,1
110 DATA 0,229,205,181,3,225,1
120 DATA 10,0,198,0,237,66,48
130 DATA 239,193,16,232,201,FIN
```

Ahora la siguiente cuestión consistirá en ver cómo añadir estos sonidos a nuestros programas. Ello supone un poco de complicación para aquellas personas que no tengan experiencia en lenguaje de máquina, lo siento. Existen tres opciones principales:

(1) Incorporar un programa cargador junto con las sentencias DATA dentro de su programa. Cuando el programa se ejecuta (utilizando SAVE LINE, sólo se tendrá que preparar la máquina una vez ya que los subsiguientes RUNs ejecutarán el programa sin activar el código de máquina nuevamente). Ello puede tomar la siguiente forma:

```
9900 REM programa cargador
...

9950 REM sentencias DATA
...

9999 GOTO 1
```

Este programa se deberá de grabar con SAVE "programa" LINE 9900.

(2)Entrar el código de máquina a mano y grabarlo en cinta utilizando SAVE CODE, después el programa que lo necesite lo puede recargar. Un ejercicio que consume tiempo.

(3)Almacenar el código en sentencias REM dentro del programa. Las sentencias REM se graban en cinta como líneas a mezclar dentro del programa que requiera los sonidos, a continuación el conjunto se puede grabar en cinta de forma que el resultado es un programa listo para funcionar contenido en el cassette. Este procedimiento es el que producirá las cargas y ejecuciones más rápidas. A continuación veremos un programa hecho con este procedimiento que contiene los doce sonidos que hemos visto. Recuerde que cada rutina es de 36 bytes de longitud y ya que hay doce, necesitaremos una sentencia REM con por lo menos 432 caracteres (12*36) tras la palabra REM en una línea. Esto significa un montón de escritura en el teclado, así que asegúrese de que sus dedos están afilados y engrasados para entrar el programa.

Las líneas de la 10 a la 60 contienen el programa cargador. El resto son sentencias DATA que contienen el código de máquina para los diversos sonidos. Naturalmente, se pueden omitir las sentencias REM ya que sólo sirven para indicar dónde comienzan los DATA para cada sonido. Observe que sólo se utiliza una variable FIN al final de toda la lista. Este programa significa mucho tecleo y hay que vigilar que todo esté correcto ya que nada funcionaría si se produce un solo error. Una vez lo haya entrado, grábelo inmediátamente en cinta ya que el más mínimo error puede hacer que el programa se pierda al intentar su ejecución.

```
   1 REM ........................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
.........................
  10 LET DIRECCIONS=PEEK 23635+2
56*PEEK 23636+5: LET FIN=1000
  20 READ BYTE
  30 IF BYTE>255 THEN STOP
  40 POKE DIRECCIONS,BYTE
  50 LET DIRECCIONS=DIRECCIONS+1
  60 GO TO 20
 100 REM BOMBA CAYENDO
 110 DATA 6,1,197,33,0,0,17,1
 120 DATA 0,229,205,161,3,1,20,0
 130 DATA 17,0,12,225,198,0,237
 140 DATA 74,229,198,0,237,82
 150 DATA 225,56,230,193,16,223
 160 DATA 201
 170 REM FUEGO DEL LASER
 180 DATA 6,1,197,33,0,0,17,1
 190 DATA 0,229,205,181,3,1,1,0
 200 DATA 17,100,1,225,198,0,237
 210 DATA 74,229,198,0,237,82
 220 DATA 225,56,230,193,16,223
 230 DATA 201
 240 REM FUEGO LASER REPETIDO
 250 DATA 6,10,197,33,0,0,17,1
 260 DATA 0,229,205,181,3,1,1,0
 270 DATA 17,100,1,225,198,0,237
 280 DATA 74,229,198,0,237,82
 290 DATA 225,56,230,193,16,223
 300 DATA 201
```

```
310 REM RASCAR
320 DATA 6,1,197,33,0,10,17,1
330 DATA 0,229,205,181,3,1,100,
0
340 DATA 17,0,30,225,198,0,237
350 DATA 74,229,198,0,237,82
360 DATA 225,56,230,193,16,223
370 DATA 201
380 REM MAQUINAS EXTRATERRESTRES
S
390 DATA 6,20,197,33,0,4,17,1
400 DATA 0,229,205,181,3,1,50,0
410 DATA 17,0,6,225,198,0,237
420 DATA 74,229,198,0,237,82
430 DATA 225,56,230,193,16,223
440 DATA 201
450 REM SONIDOS MISTERIOSOS
460 DATA 6,5,197,33,0,10,17,10
470 DATA 0,229,205,181,3,1,0,2
480 DATA 17,0,19,225,198,0,237
490 DATA 74,229,198,0,237,82
500 DATA 225,56,230,193,16,223
510 DATA 201
520 REM ALARMA
530 DATA 6,10,197,33,0,0,17,100
540 DATA 0,229,205,181,3,1,0,1
550 DATA 17,0,3,225,198,0,237
560 DATA 74,229,198,0,237,82
570 DATA 225,56,230,193,16,223
580 DATA 201
590 REM DESTELLO DEL LASER
600 DATA 6,25,197,33,0,0,17,6
610 DATA 0,229,205,181,3,1,50,0
620 DATA 17,0,1,225,198,0,237
630 DATA 74,229,198,0,237,82
640 DATA 225,56,230,193,16,223
650 DATA 201
660 REM SINTETIZADOR 'PEEOW'
670 DATA 6,1,197,33,0,0,17,5
680 DATA 0,229,205,181,3,1,1,0
690 DATA 17,0,1,225,198,0,237
700 DATA 74,229,198,0,237,82
710 DATA 225,56,230,193,16,223
720 DATA 201
730 REM PAJARO
740 DATA 6,14,197,33,0,0,17,40
750 DATA 0,229,205,181,3,1,25,0
760 DATA 17,240,0,225,198,0,237
770 DATA 74,229,198,0,237,82
780 DATA 225,56,230,193,16,223
790 DATA 201
800 REM TONO ASCENDENTE
810 DATA 6,1,197,33,232,3,17,1
820 DATA 0,229,205,181,3,225,1
830 DATA 1,0,198,0,237,66,48
840 DATA 239,193,16,232,201
850 DATA 0,0,0,0,0,0,0,0,0
860 REM TONO ASCENDENTE RAPIDO
870 DATA 6,1,197,33,232,2,17,1
880 DATA 0,229,205,181,3,225,1
```

```
890 DATA 10,0,198,0,237,66,48
900 DATA 239,193,16,232,201
910 DATA 201,FIN
```

Tras grabar el programa anterior en la cinta, ejecútelo (RUN) para activar el código de máquina, ello tomará algunos segundos y finalmente se detendrá con un 9 STOP 30:2. Para comprobar si todo está correcto, utilice el comando PRINT ADDRESS, que debe de dar un valor 24183. Esto trabaja solamente si Vd. no tiene enchufado ningún módulo de expansión, etc.,ya que entonces el programa comenzará en una dirección diferente. La comprobación final consiste en ejecutar el código de máquina (por esto es por lo que le he dicho que salve el programa en cinta una vez lo haya tecleado). Introduzca el siguiente programa de comprobación antes de borrar cualquier línea:

```
  10 LET DIRECCION=PEEK 23635+25
6*PEEK 23636+5
  11 FOR A=DIRECCION TO DIRECCIO
N+423 STEP 36
  12 RANDOMIZE USR A
  13 PAUSE 20
  14 NEXT A: STOP
```

Si todo está correcto, se deberán de escuchar los doce sonidos,separados,cada uno de ellos,por una pausa y el programa se detendrá normalmente en la línea 14. Si todo está correcto, Vd. puede seguir adelante y borrar todas las líneas excepto la 1 REM y salvar esta línea en la cinta. Esta línea es la que Vd. añadirá a sus futuros programas de juegos. Ahora lo único que resta es invocar cada sonido conforme se necesite. Todos aquellos que no tengan módulo de expansión, no saben cuán felices son ya que pueden llamar a la dirección apropiada debido a que el REM queda fijado en la memoria. La tabla siguiente indica las direcciones a invocar.

NUMERO DEL SONIDO	USR ADDRESS	SONIDO
1	23760	Estallido de bomba
2	23796	Disparo laser
3	23832	Disparo laser repetido
4	23868	Rasgadora
5	23904	Platillo volante
6	23940	Sonidos miteriosos
7	23976	Alarma
8	24012	Rayo laser
9	24048	Tambor
10	24084	Pájaro (o viendo las estrellas!)
11	24120	Tono ascendente
12	24156	Tono ascendente rápido

Esta tabla de direcciones probablemente no funcionará con aquellos Spectrums que tengan mapas de microdrive, etc., por debajo del área de programas. Por ello le sugiero que invoque a la rutina apropiada mediante PEEK 23635/6 y añadiendo al resultado el número apropiado de bytes. Este trabajo resulta bastante tedioso, por lo que le sugiero que utilice una FN para que haga los cálculos por Vd.:

DEF FN U(N)=PEEK 23635+256*PEEK 23636+5+36*(N–1)

Inclúyala como línea 2 y salve el programa junto con la línea REM. Cada vez que Vd. desee escuchar uno de los sonidos (por ejemplo, el sexto), haga lo siguiente:

LET U=USR FN U(6)

Debido a que los sonidos se utilizan normalmente en juegos, es mejor asignar el número retornado por la función de una llamada USR a una variable en vez de utilizar RANDOMIZE USR, ya que ello afectará a los números aleatorios generados por el RANDOM. A continuación veamos un ejemplo de cómo utilizar USR FN:

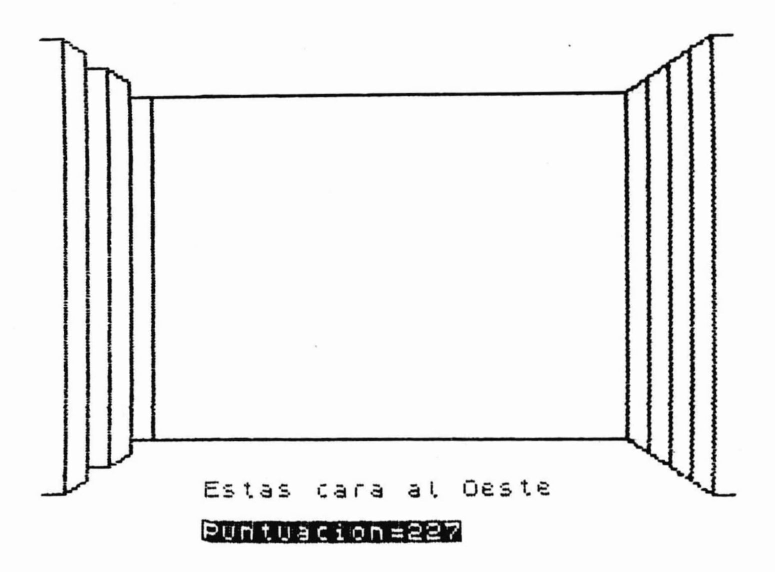
```
   2 DEF FN u(n)=PEEK 23635+256*
PEEK 23636+5+36*(n-1)
   3 REM SONIDO N CON USR FN(N)
  10 FOR A=1 TO 12
  20 LET U=USR FN U(A)
  30 NEXT A
```

He aquí un programa gráfico de ejemplo que muestra cómo utilizar los sonidos. Suena una alarma cada vez que se acerca un platillo volante. En su primer paso destruye una base dejando caer una bomba sobre ella y en el segundo paso, la segunda base dispara. Un ejemplo bastante violento pero representa una útil demostración de los sonidos. Funciona permanentemente hasta que se corte con BREAK. Espero que todo esto le animará a experimentar en la creación de sus propios sonidos.

```
   2>DEF FN U(N)=PEEK 23635+256*
PEEK 23636+5+36*(N-1)
   3 REM SOUND N WITH USR FN(N)
 100 PRINT AT 15,12;"▄▄█▄▄";AT 1
5,20;"▄▄█▄▄": REM BASE
 105 PRINT AT 0,8; FLASH 1;"UFO
APPROACHING"
 110 LET R=USR FN U(7): REM ALAR
M
 115 PRINT AT 0,8;"
 "
 120 FOR A=0 TO 30
 130 PRINT AT 5,A;" >"
 140 IF A<5 THEN GO TO 180: REM
NEXT
 150 IF A=5 THEN LET R=USR FN U(
12): PRINT AT 6,5;"|": REM DROP
BOMB
 160 IF A>5 AND A<14 THEN PRINT
AT A,A-1;" ";AT A+1,A;"|"
 170 IF A=14 THEN PRINT AT A,A-1
;" ";AT 15,12; FLASH 1; OVER 1;"
     ": LET R=USR FN U(1): PRINT
 AT 15,12;"     "
 180 NEXT A
 185 PRINT AT 5,31;" "
 186 PRINT AT 0,8; FLASH 1;"UFO
APPROACHING"
 187 LET R=USR FN U(7): REM ALAR
M
 188 PRINT AT 0,8;"
 "
 190 FOR A=0 TO 21
 200 PRINT AT 5,A;" >"
 210 NEXT A
 220 PLOT 179,56: DRAW 0,75
 225 LET R=USR FN U(8)
 230 PLOT OVER 1;179,56: DRAW OV
ER 1;0,75
 240 PRINT AT 5,22; FLASH 1;">"
 250 LET R=USR FN U(4)
 255 PAUSE 50
 260 CLS : GO TO 100
```

LABERINTO TRIDIMENSIONAL (16K)

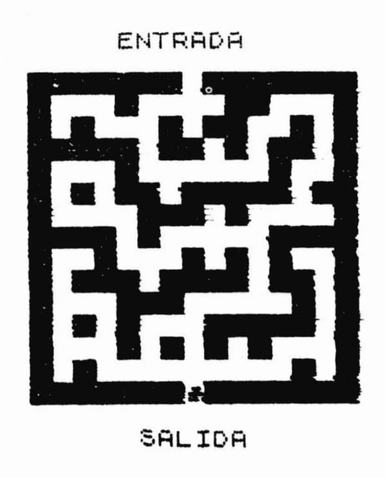
¿Qué tal un paseo por dentro de un laberinto en el cual Vd. puede ver los muro: y corredores en un glorioso efecto tridimensional?. Si Vd. se pierde por dentr(de un corredor sin final, consulte a su computador para que le ayude y le mues tre un mapa del laberinto a costa de una pérdida de 20 puntos de los 300 de qu(dispone Vd. al empezar el juego. El objeto del programa es, por supuesto, sali del laberinto con la máxima puntuación posible. La puntuación se rebaja a cad: movimiento y se descuentan 20 puntos cada vez que se pide ayuda. Una vez Vd domine este laberinto, puede cambiar el diseño del mismo.



En cualquier caso comience por entrar el programa, que está pensado para un máquina de 16K. Recuerde que no hay diferencia entre los nombres de variabl en mayúsculas y minúsculas, de modo que LABEY es exactamente lo mism que labey, por ejemplo. Utilice los números de línea al pie de la letra ya que ha varios GOSUBs y GOTOs calculados. Donde vea referencias a D$, cualquie valor "N" (Norte), "S" (Sur), "E" (Este) y "O" (Oeste), se deben de poner e mayúsculas ya que el programa utiliza esta variable para saber en qué direcció va Vd. y sólo comprueba las mayúsculas. También existen varias líneas de sen tencias múltiples IF...THEN... Se utilizan cuando hay más de una posibilida para una cierta condición pero sin que sea realmente necesaria una subrutin separada.

En la línea 8015 hay cuatro apóstrofes en la sentencia PRINT. En las líneas 8510 y 8515 se utiliza el símbolo de barras cruzadas (el mismo del juego de tres en raya) en la tecla tres. Los caracteres DATA usados en el laberinto son cuadrados gráficos SHIFT 8, pero pueden ser cualquiera que Vd. prefiera ya que el programa sólo comprueba la presencia de un espacio para denotar un corredor abierto, cualquier otro carácter significa un muro.

Una vez se ha introducido todo el programa, grábelo en cinta y verifíquelo. Entonces ya está listo para ejecutarlo. La pantalla cambiará a blanco sobre azul con un borde negro. Sonará un tono de dos notas una vez el laberinto esté a punto apareciendo el mensaje "pulsa una tecla para comenzar". Una vez haya pulsado la tecla, aparecerá un mapa en el laberinto durante unos dos segundos con la entrada marcada "ENTRADA" y la salida "SALIDA". Un mensaje al fondo de la pantalla indica en qué dirección está Vd. mirándolo así como la puntuación inicial que tiene. La posición suya se indica por un asterisco, que en esta ocasión está en la entrada.



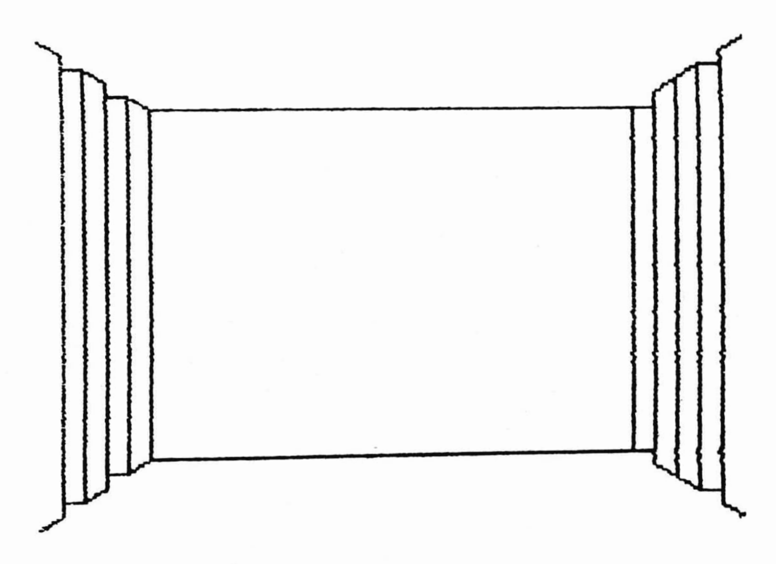
You are facing north 

A continuación se borrará la pantalla dejándolo a Vd. solo dentro del laberinto. Al frente verá una representación tridimensional de la entrada del laberinto. Los controles de que Vd. dispone son:

5: Girar a la izquierda (90 grados en sentido opuesto a las agujas del reloj).

6: Marcha atrás (girar 180 grados).
7: Avanzar (el espacio correspondiente a un cuadro del mapa).
8: Girar a la derecha (90 grados en sentido de las agujas del reloj).
h o H: ¡¡Socorro!!.
o Ya me he cansado..., quiero acabar..., sacarme de aquí...

La última opción es realmente la *última,* ya que todo lo que hace es parar el programa. Utilícela en vez del BREAK, ya que puede que haya alguna dirección de retorno en el stack de RETURNs y se puede colgar la memoria de la máquina. Utilice la opción F para suspender el programa si Vd. tiene que atender el teléfono o ir al lavabo. Para proseguir con el juego, use CONTINUE. No se obtendrá indicación de la dirección en que se avanza hasta que se efectúe un movimiento, ya que el mensaje STOP la habrá borrado de la pantalla.



Estas cara al Norte

Puntuacion=300

```
   1 REM Laberinto 3D
   3 REM por
   4 REM DILWYN JONES
  10 BRIGHT 0: FLASH 0: INVERSE
0: OVER 0
  20 INK 7: PAPER 1: BORDER 0: C
LS
  30 GO SUB 9000
 100 REM TECLADO
```

N. del T. Cuando aparezca " ", entrar el espacio con el 8 en modo gráfico o bien el 143 del Código ASC11.

```
 110 LET K$=INKEY$
 112 IF K$="H" OR K$="h" THEN GO
 SUB 8000: GO SUB 8500
 113 IF K$="q" OR K$="Q" THEN ST
OP
 115 IF (K$<"5" OR K$>"8") THEN
GO TO 110
 117 LET score=score-1
 120 IF K$="5" OR K$="6" OR K$="
8" THEN CLS : GO SUB 1200: GO SU
B 8500
 130 IF K$="7" THEN GO SUB 1000:
 GO SUB 8500
 140 IF LABEY=SALIRY AND LABEX=S
ALIRX THEN PRINT AT 1,3;"SALISTE
! "; FLASH 1; BRIGHT 1;score: PR
INT AT 1,18;"Puntos" : STOP
 200 GO TO 100
 230 STOP
1000 REM Movimiento adelante
1005 IF (D$="N" AND LABEY=1) OR
(D$="S" AND LABEY=15) OR (D$="E"
 AND LABEX=15) OR (D$="W" AND LA
BEX=1) THEN RETURN
1010 IF D$="N" THEN IF M$(LABEY-
1,LABEX)<>" " THEN RETURN
1020 IF D$="S" THEN IF M$(LABEY+
1,LABEX)<>" " THEN RETURN
1030 IF D$="O" THEN IF M$(LABEY,
LABEX-1)<>" " THEN RETURN
1040 IF D$="E" THEN IF M$(LABEY,
LABEX+1)<>" " THEN RETURN
1050 LET LABEX=LABEX+(D$="E" AND
 LABEX<=15)-(D$="O" AND LABEX>=1
)
1060 LET LABEY=LABEY+(D$="S" AND
 LABEY<=15)-(D$="N" AND LABEY>=1
)
1070 RETURN
1200 REM Girar
1210 IF K$="5" THEN LET D$=("O"
AND D$="N")+("S" AND D$="O")+("E
" AND D$="S")+("N" AND D$="E")
1220 IF K$="8" THEN LET D$=("E"
AND D$="N")+("N" AND D$="O")+("O
" AND D$="S")+("S " AND D$="E"
)
1230 IF K$="6" THEN LET D$=("S"
AND D$="N")+("E" AND D$="O")+("N
" AND D$="S")+("O" AND D$="E")
1235 CLS
1240 GO TO 2000+(200 AND D$="O")
+(400 AND D$="N")+(600 AND D$="S
")
2000 REM Cara este
2010 FOR m=0 TO 16-LABEX
2020 LET X=8*m
2025 IF LABEY=15 THEN GO SUB 700
0: GO SUB 4000+(2000 AND M$(LABE
Y-1,LABEX-M)=" "): GO TO 2070
```

```
2026 IF LABEY=15 THEN GO SUB 700
0: GO SUB 4000+(2000 AND M$(LABE
Y+1,LABEX+M)=" "): GO TO 2070
2030 GO SUB 4000+(2000 AND M$(LA
BEY-1,LABEX+M)=" ")
2050 GO SUB 5000+(2000 AND M$(LA
BEY+1,LABEX+M)=" ")
2070 IF LABEX+M+1<=15 THEN IF M$
(LABEY,LABEX+M+1)<>" " THEN GO T
O 3500
2080 NEXT M
2090 RETURN
2200 REM Cara Oeste
2210 FOR m=0 TO LABEX-1
2220 LET X=8*M
2225 IF LABEY=15 THEN GO SUB 600
0: GO SUB 5000+(2000 AND M$(LABE
Y-1,LABEX-M)=" "): GO TO 2270
2226 IF LABEY=1 THEN GO SUB 7000
: GO SUB 4000+(2000 AND M$(LABEY
+1,LABEX-M)=" "): GO TO 2270
2230 GO SUB 4000+(2000 AND M$(LA
BEY+1,LABEX-M)=" ")
2250 GO SUB 5000+(2000 AND M$(LA
BEY-1,LABEX-M)=" ")
2270 IF LABEX-M-1>=1 THEN IF M$(
LABEY,LABEX-M-1)<>" " THEN GO TO
 3500
2280 NEXT M
2290 RETURN
2400 REM Cara Norte
2410 FOR M=0 TO LABEY-1
2420 LET X=M*8
2425 IF LABEX=15 THEN GO SUB 700
0: GO SUB 4000+(2000 AND M$(LABE
Y-M,LABEX-1)=" "): GO TO 2470
2426 IF LABEX=1 THEN GO SUB 6000
: GO SUB 5000+(2000 AND M$(LABEY
-M,LABEX+1)=" "): GO TO 2470
2430 GO SUB 4000+(2000 AND M$(LA
BEY-M,LABEX-1)=" ")
2450 GO SUB 5000+(2000 AND M$(LA
BEY-M,LABEX)=" ")
2470 IF LABEY-M-1>=1 THEN IF M$(
LABEY-M-1,LABEX)<>" " THEN GO TO
 3500
2480 NEXT M
2490 RETURN
2600 REM Cara al Sur
2610 FOR M=0 TO 15-LABEY
2620 LET X=8*M
2625 IF LABEX=15 THEN GO SUB 600
0: GO SUB 5000+(2000 AND M$(LABE
Y+M,LABEX-1)=" "): GO TO 2670
2626 IF LABEX=1 THEN GO SUB 7000
: GO SUB 4000+(2000 AND M$(LABEY
+M,LABEX+1)=" "): GO TO 2670
2630 GO SUB 4000+(2000 AND M$(LA
BEY+M,LABEX+1)=" ")
```

```
2650 GO SUB 5000+(2000 AND M$(LA
BEY+M,LABEX-1)=" ")
2670 IF LABEY+M+1<=15 THEN IF M$
(LABEY+M+1,LABEX)<>" " THEN GO T
O 3500
2680 NEXT M
2690 RETURN
3500 REM Pared delante
3510 LET x=x+8
3520 PLOT x,x*5/8: DRAW 255-(2*x
),0
3530 PLOT x,x*5/8+175-(10*x/8):
DRAW 255-(2*x),0
3540 RETURN
4000 REM Bloqueo pared Izquierda
4010 PLOT X,X*5/8
4020 DRAW 8,5
4030 DRAW 0,165-(10*X/8)
4040 DRAW -8,5
4050 RETURN
5000 REM Dibujar pared derecha
5010 PLOT 255-x,x*5/8
5020 DRAW -8,5
5030 DRAW 0,165-(5*x/4)
5040 DRAW 8,5
5050 RETURN
6000 REM Abrir Corredor Iz.
6010 PLOT X,X*5/8+5
6020 DRAW 8,0
6030 DRAW 0,165-(5*X/4)
6040 DRAW -8,0
6050 RETURN
7000 REM Abrir Corredor Der.
7010 PLOT 255-X,X*5/8+5
7020 DRAW -8,0
7030 DRAW 0,165-(5*X/4)
7040 DRAW 8,0
7050 RETURN
8000 REM SOCORRO!
8010 CLS
8015 PRINT ''''
8020 FOR A=1 TO 15
8030 PRINT TAB 8;M$(A)
8040 NEXT A
8050 PRINT AT LABEY+3,LABEX+7;"*
"
8051 IF SALIRY=1 OR SALIRY=15 TH
EN PRINT AT 2+(18 AND SALIRY=15)
,SALIRX+6;"FUERA"
8052 IF SALIRY>1 AND SALIRY<15 T
HEN PRINT AT SALIRY+3,4+(20 AND
SALIRX=15);"SALIR"
8053 IF ENTRARY=1 OR ENTRARY=15
THEN PRINT AT 2+(18 AND ENTRARY=
15),ENTRARX+6;"DENTRO"
8054 IF ENTRARY>1 AND ENTRARY<15
 THEN PRINT AT ENTRARY+3,5+(19 A
ND ENTRARX=15);"DENTRO"
8059 LET score=score-20: GO SUB 8500
```

```
8500
8060 PAUSE 100: CLS
8070 GO SUB 2000+(200 AND D$="W"
)+(400 AND D$="N")+(600 AND D$="
S")
8090 RETURN
8500 REM Que Direccion?
8505 IF score<0 THEN LET score=0
8510 PRINT #1;AT 1,0;"Vas hacia
el ";"Norte" AND D$="N";"Sur" AN
D D$="S";"Este " AND D$="E";"Oes
te " AND D$="O"
8515 PRINT #1;AT 1,24-LEN STR$ s
core; INVERSE 1;"Puntos=";score
8520 RETURN
9000 REM INICIALIZACION
9020 DATA "               "
9030 DATA "               "
9040 DATA "               "
9050 DATA "               "
9060 DATA "               "
9070 DATA "               "
9080 DATA "               "
9090 DATA "               "
9100 DATA "               "
9110 DATA "               "
9120 DATA "               "
9130 DATA "               "
9140 DATA "               "
9150 DATA "               "
9160 DATA "               "
9170 LET ENTRARX=8: LET ENTRARY=
15
9180 LET SALIRX=8: LET SALIRY=1
9190 DIM M$(15,15): REM LABERINT
O
9200 RESTORE 9020
9210 FOR M=1 TO 15
9220 READ M$(M)
9230 NEXT M
9240 LET LABEX=ENTRARX
9250 LET LABEY=ENTRARY
9260 LET D$="N"
9261 BEEP .5,0: BEEP .7,12
9262 IF INKEY$<>"" THEN GO TO 92
62
9263 PRINT "Pulsa una tecla para
 empezar"
9264 PAUSE 0
9265 LET score=320
9266 GO SUB 8000
9267 GO SUB 8500
9270 GO SUB 2400
9290 GO SUB 8500
9300 RETURN
```

El listado es bastante largo y complejo. Existe un montón de repeticiones de las rutinas de dibujo con el fin de evitar una multitud de sentencias condiciona-

les que si se pusieran, eventualmente acortarían el programa pero lo harían más lento. Tal como está el programa, puede sorprenderle agradablemente el ver como un programa en BASIC de esta complejidad puede ejecutarse tan rápido habiendo como hay un montón de trabajo para obtener las vistas en perspectiva y dibujarlas. El dibujo no es estrictamente preciso en cuanto a que las alturas deben de cambiar con la distancia, y el ancho de las entradas a los corredores no aumenta conforme Vd. se acerca a las mismas. Esto se podría conseguir pero no solamente haría el programa más lento, hasta límites inaceptables, sino que también las entradas a los corredores más lejanos aparecerían tan finas que no se podrían distinguir. Como puede ver en el ejemplo, el método utilizado para dibujar da un efecto tridimensional más que aceptable ya que se pueden ver las paredes a izquierda y derecha, distinguiéndose netamente las entradas a los corredores laterales.

Las líneas 10 y 20 colocan los colores, etc. Consisten sólo en comando de color global. De forma que si el texto en blanco y los gráficos son de un fondo azul, no le gustan, Vd. puede cambiar estas sentencias. El borde negro se utiliza para encuadrar la vista y a algunos puede no gustarle; nuevamente, si Vd. cambia la sentencia BORDER utilizada en el programa, puede colorearlo a su gusto. La línea 30 envía el programa a la rutina de inicialización de la línea 9000. El bucle de las líneas 100 a la 200 negocia con la lectura del teclado tomando la acción adecuada. La línea 112 maneja la opción de "Socorro" yendo a las rutinas de la línea 8000 (ayuda) y 8500 (dirección/puntuación). La rutina de ayuda deduce la cantidad apropiada de la puntuación, y la rutina de dirección/puntuación imprime ambos datos en la pantalla. La línea 113 es para la opción de fin de juego. La línea 115 asegura que el programa no avanza más en el caso de que no se pulse ninguna tecla o de que se haya pulsado la opción de ayuda. Esto hace que la puntuación trabaje correctamente. La línea 117 decrementa la puntuación en una unidad a cada movimiento que se haga, excepto en el caso de ayuda, en el cual se toma una acción diferente. La línea 120 trabaja con las opciones de giro borrando primero la pantalla y llamando a continuación a la subrutina de giro de la línea 1200 y, seguidamente, a la subrutina de dirección/puntuación. La línea 130 controla el avance llamando a la subrutina 1000 y a la de dirección-puntuación. La línea 140 comprueba si Vd. ha llegado al final del laberinto, imprimiendo la puntuación y deteniendo el programa. En la línea 1000 se inicia la subrutina de avance. La línea 1005 comprueba los límites del laberinto. Las líneas de la 1010 a la 1040 comprueban si hay una obstrucción al frente. Observe la utilización de IF...THEN IF...THEN en lugar de IF...AND...THEN... ya que el segundo procedimiento puede causar un error bajo ciertas condiciones tales como intentar girar en la misma entrada del laberinto. El programa salta sobre todas ellas sin intentar ejecutar nada que pudiera causar un error. Es una característica del BASIC del Spectrum ZX el que cualquier cosa que venga a continuación del THEN se salta hasta el final de la línea salvo que la condición sea verdad. Las líneas 1050 y 1060 cambian los valores de las varia-

bles LABEY y LABEX de la forma apropiada. Estas dos variables determinan la posición X e Y dentro del laberinto respectivamente. La línea 1200 es el inicio de la rutina de giro, la cual cambia a la variable de dirección D$ tal como sea necesario. La línea 1240 elige la rutina de dibujo a utilizar dependiendo de la dirección de avance, mediante un GOTO calculado, el mismo que se utiliza para avanzar.

Las líneas de la 2000 a la 2090 forman la rutina que manejan el dibujo cuando se mira hacia el este y es similar a las otras tres rutinas de dirección que llegan hasta la línea 2690. El bucle m trabaja desde su posición actual al límite exterior del laberinto. X es donde estará la coordenada a través de la pantalla desde la parte más cercana del final hasta Vd. correspondiendo a la pared que se dibujará a la izquierda de la pantalla. X se utilizará como factor de escala y también como vista de la perspectiva. Las líneas 2025 a 2050 seleccionan cuando se debe de dibujar una pared abierta o una cerrada a la izquierda o a la derecha. La línea 2070 determina si hay o no hay una obstrucción delante de Vd., en cuyo caso se dibuja una pared al frente en la línea 3500. Las rutinas de las líneas 4000 a la 7050 negocian con el dibujo en sí mismo de las paredes a ambos lados utilizando una anchura de 8 pixels para cada pared/abertura de corredor (y con el hecho de que cada pared sucesiva será 12 pixels más corta ya que se encuentran más lejos de Vd.). Esta es la razón por la cual el tamaño del laberinto se ha fijado a 15 por 15. La línea 8000 es el comienzo de la rutina de ayuda, la cual imprime el mapa del laberinto, mostrando su posición mediante un asterisco. Las líneas 8051 a 8054 determinan donde marcar la entrada y la salida y que se pueden cambiar como se describirá más tarde. La línea 8059 deduce los puntos de penalización por la demanda de socorro. La línea 8060 determina de cuánto tiempo se dispone para estudiar el mapa del laberinto antes de borrar la pantalla y continuar el programa dibujando la posición actual del laberinto (línea 8070).

La línea 8500 imprime su puntuación y dirección (de la 8510 a la 8515) en la parte baja de la pantalla utilizando PRINT1 para hacer que aparezca como si estuviera escrito en el área del borde, fuera de la zona principal. La línea 8505 asegura de que la puntuación nunca bajará de 0. Finalmente llegamos a la línea 9000 que es donde comienza la rutina principal de inicialización. Las líneas 9020 a la 9180 se deben introducir siguiendo exactamente los números del listado en el caso de que posteriormente Vd. desee cambiar el diseño del laberinto.

Las sentencias DATA están colocadas de forma que el diseño del laberinto sea obvio al ver el listado. ENTRARX es la coordenada transversal desde la entrada (de 1 a 15, de izquierda a derecha). ENTRARY es la coordenada de la entrada Y (1 a 15 de arriba abajo). SALIRX es la coordenada de la salida X y SALIRY es la coordenada Y de la salida, como implica el nombre de la variable. La línea 9190 prepara la matriz M$ (15,15) que contiene el laberinto. Esta matriz se puede

salvar en cinta utilizando SAVE"nombre"DATA pero, naturalmente, se habrá borrado si se ha utilizado un RUN, es por ello que utilizaremos sentencias DATAs para diseñar nuevos laberintos. Las líneas 9210 a la 9230 leen el laberinto de las sentencias DATA y lo colocan en la matriz, a continuación las coordenadas de inicio se colocan de modo que correspondan a las de la entrada. D$ se coloca al valor "N" lo que indica que el juego comienza mirando al norte en este caso.

La línea 9261 genera un tono corto de dos notas para avisar que todo está a punto. La línea 9262 queda a la espera de la pulsación de las teclas. Las líneas 9263/4 esperan a que Vd. pulse cualquier tecla (incluso los SHIFTs). La puntuación inicial se coloca a 320 ya que la rutina de ayuda se utiliza para la presentación inicial del mapa y deducirá 20 puntos dejándolo en 300 que es lo que nosotros deseamos. Las líneas 9266 a 9290 preparan la visualización y la dirección /puntuación dibujando seguidamente la entrada tridimensional del laberinto.

Cambios en este programa

El cambio más fácil consiste en modificar los colores de la pantalla, que están en blanco sobre fondo azul con un borde negro. Las líneas 10 y 20 contienen todos los elementos necesarios para el cambio.

Se puede cambiar el tiempo de presentación del mapa modificando la sentencia PAUSE de la línea 8060. El carácter usado para mostrar su posición puede ser cualquiera que Vd. desee, incluso un espacio, pero si coloca un espacio jamás sabrá dónde se encuentra Vd. dentro del laberinto.

Se puede cambiar la puntuación inicial en la línea 9265 en el caso de que a Vd. le sea muy dificil salir del laberinto. Y si desea restar más o menos puntos al pedir ayuda, modifique la línea 8059. También, si Vd. está más inclinado musicalmente que yo, puede hacer una rutina musical que suene cuando se sale del laberinto. Esto se puede conseguir reemplazando el STOP de la línea 140 con un GOTO a una rutina que toque la música. Esta rutina se puede colocar entre las líneas 200 y 1000 ya que esta parte no se utiliza, estando situada entre las subrutinas y después del bucle principal.

El mayor cambio que se puede hacer es el laberinto en si. Lo mejor consiste en salvar las líneas 9020 a 9180 por separado en la cinta y luego mezclarlas con el programa principal de forma que se pueda cargar fácilmente un nuevo laberinto.

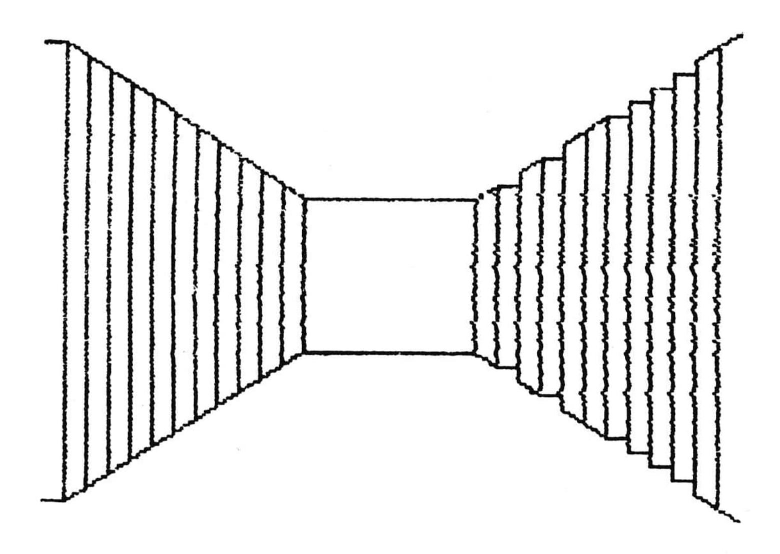
He aquí un ejemplo:

```
9020>DATA ""
9030 DATA ""
9040 DATA ""
9050 DATA ""
9060 DATA ""
9070 DATA ""
9080 DATA ""
9090 DATA ""
9100 DATA ""
9110 DATA ""
9120 DATA ""
9130 DATA ""
9140 DATA ""
9150 DATA ""
9160 DATA ""
9170 LET ENTRARX=15: LET ENTRARY
=3
9175 LET D$="O"
9180 LET SALIRX=1: LET SALIRY=13
```



```
9020>DATA ""
9030 DATA ""
9040 DATA ""
9050 DATA ""
9060 DATA ""
9070 DATA ""
9080 DATA ""
9090 DATA ""
9100 DATA ""
9110 DATA ""
9120 DATA ""
9130 DATA ""
9140 DATA ""
9150 DATA ""
9160 DATA ""
9170 LET ENTRARX=1: LET ENTRARY=
7
9175 LET D$="E"
9180 LET SALIRX=1: LET SALIRY=3
```

Algo que hay que vigilar es no colocar corredores de más de un cuadrado de ancho. Tal como puede ver en este ejemplo en el cual el primer corredor a la derecha consistiría en un pasillo de cuatro veces el ancho normal de un corredor, esto aparece en la pantalla como cuatro corredores independientes colocados uno a continuación del otro. Es correcto del todo pero inicialmente puede inducir a confusión.



Si Vd. dispone de suficiente memoria, puede almacenar varios laberintos en sentencias DATAs consecutivas al final del programa. Los valores de ENTRARX, ENTRARY, D$, SALIRX y SALIRY también se deben de mantener dentro de las sentencias DATA. Se utilizará la sentencia RESTORE (número de línea) para seleccionar el laberinto a utilizar. Todo esto puede permitirle tener tantos laberintos como quepan en la memoria de su computador.

El ZX Spectrum es un ordenador que comparado con otros del mismo estilo resulta mucho más fácil de manejar.

Por esto, en este libro, se enseña de una manera clara y sencilla, como el aparato mismo, a profundizar en el ZX Spectrum.

Esta obra no quiere ser un libro de texto sino que está concebido para ser el compañero inseparable en sus exploraciones a través del Spectrum.

Tanto si quiere profundizar en el ROM como si quiere divertirse con un juego en tres dimensiones, aquí encontrará toda la información que necesite.

EDITORIAL NORAY
San Gervasio de Cassolas, 79
BARCELONA